



BlackBerry Java Development Environment

Version 4.3.0

Development Guide

BlackBerry Java Development Environment Version 4.3.0 Development Guide

Last modified: 19 September 2007

Part number: 12083107

At the time of publication, this documentation is based on the BlackBerry Java Development Environment Version 4.3.0.

Send us your comments on product documentation: <https://www.blackberry.com/DocsFeedback>.

©2007 Research In Motion Limited. All Rights Reserved. The BlackBerry and RIM families of related marks, images, and symbols are the exclusive properties of Research In Motion Limited. RIM, Research In Motion, BlackBerry, "Always On, Always Connected" and the "envelope in motion" symbol are registered with the U.S. Patent and Trademark Office and may be pending or registered in other countries.

Bluetooth is a trademark of Bluetooth SIG. Java and JavaScript are trademarks of Sun Microsystems, Inc. Microsoft, ActiveX, Internet Explorer, and Windows are trademarks of Microsoft Corporation. iCal is a trademark of Apple Computer, Inc. vCard is a trademark of the Internet Mail Consortium. Plazmic is a trademark of Plazmic Inc. Wi-Fi and 802.11, 802.11a, 802.11b, 802.11g are trademarks of the Wi-Fi Alliance. All other brands, product names, company names, trademarks and service marks are the properties of their respective owners.

The BlackBerry device and/or associated software are protected by copyright, international treaties and various patents, including one or more of the following U.S. patents: 6,278,442; 6,271,605; 6,219,694; 6,075,470; 6,073,318; D445,428; D433,460; D416,256. Other patents are registered or pending in various countries around the world. Visit www.rim.com/patents.shtml for a list of RIM (as hereinafter defined) patents.

This document is provided "as is" and Research In Motion Limited and its affiliated companies ("RIM") assume no responsibility for any typographical, technical or other inaccuracies in this document. In order to protect RIM proprietary and confidential information and/or trade secrets, this document may describe some aspects of RIM technology in generalized terms. RIM reserves the right to periodically change information that is contained in this document; however, RIM makes no commitment to provide any such changes, updates, enhancements or other additions to this document to you in a timely manner or at all. RIM MAKES NO REPRESENTATIONS, WARRANTIES, CONDITIONS OR COVENANTS, EITHER EXPRESS OR IMPLIED (INCLUDING WITHOUT LIMITATION, ANY EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS OF FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, MERCHANTABILITY, DURABILITY, TITLE, OR RELATED TO THE PERFORMANCE OR NON-PERFORMANCE OF ANY SOFTWARE REFERENCED HEREIN OR PERFORMANCE OF ANY SERVICES REFERENCED HEREIN). IN CONNECTION WITH YOUR USE OF THIS DOCUMENTATION, NEITHER RIM NOR ITS RESPECTIVE DIRECTORS, OFFICERS, EMPLOYEES OR CONSULTANTS SHALL BE LIABLE TO YOU FOR ANY DAMAGES WHATSOEVER BE THEY DIRECT, ECONOMIC, COMMERCIAL, SPECIAL, CONSEQUENTIAL, INCIDENTAL, EXEMPLARY OR INDIRECT DAMAGES, EVEN IF RIM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, INCLUDING WITHOUT LIMITATION, LOSS OF BUSINESS REVENUE OR EARNINGS, LOST DATA, DAMAGES CAUSED BY DELAYS, LOST PROFITS, OR A FAILURE TO REALIZE EXPECTED SAVINGS.

This document might contain references to third party sources of information, hardware or software, products or services and/or third party web sites (collectively the "Third-Party Information"). RIM does not control, and is not responsible for, any Third-Party Information, including, without limitation the content, accuracy, copyright compliance, compatibility, performance, trustworthiness, legality, decency, links, or any other aspect of Third-Party Information. The inclusion of Third-Party Information in this document does not imply endorsement by RIM of the Third Party Information or the third party in any way. Installation and use of Third Party Information with RIM's products and services may require one or more patent, trademark or copyright licenses in order to avoid infringement of the intellectual property rights of others. Any dealings with Third Party Information, including, without limitation, compliance with applicable licenses and terms and conditions, are solely between you and the third party. You are solely responsible for determining whether such third party licenses are required and are responsible for acquiring any such licenses relating to Third Party Information. To the extent that such intellectual property licenses may be required, RIM expressly recommends that you do not install or use Third Party Information until all such applicable licenses have been acquired by you or on your behalf. Your use of Third Party Information shall be governed by and subject to you agreeing to the terms of the Third Party Information licenses. Any Third Party Information that is provided with RIM's products and services is provided "as is". RIM makes no representation, warranty or guarantee whatsoever in relation to the Third Party Information and RIM assumes no liability whatsoever in relation to the Third Party Information even if RIM has been advised of the possibility of such damages or can anticipate such damages.

Research In Motion Limited
295 Phillip Street
Waterloo, ON N2L 3W8
Canada

Published in Canada

Research In Motion UK Limited
Centrum House, 36 Station Road
Egham, Surrey TW20 9LF
United Kingdom

Contents

1	Creating user interfaces	15
	Elements of a BlackBerry device user interface	15
	Screens	15
	UI components	16
	Create a screen	17
	Adding UI components to a screen	17
	Create UI components	17
	Creating custom UI components	20
	Create a custom field	20
	Create custom context menus	25
	Create custom layout managers	26
	Create custom lists	28
	Adding menu items to BlackBerry Java Applications	29
	Create a menu item	29
	Register a menu item	30
	Arrange UI components	30
	Define a layout manager	30
	Set field focus and navigation	30
	Listen for field focus changes	31
	Respond to UI events	31
	Listen for field property changes	33
	Manage foreground events	34
	Manage drawing areas	34
	Code samples	35
	Code sample: Creating custom buttons	35
	Code sample: Creating a custom context menu	38
	Code sample: Creating a custom layout manager	40
	Code sample: Creating a custom list	41
	Code sample: Creating a new menu item in a BlackBerry Java Application	42
2	Using graphics and multimedia	45
	Using images	45
	Use raw images	45

Use encoded images	46
Drawing and rendering images	47
Position an image	47
Draw an image in color	47
Using audio.....	48
Start the media player from the BlackBerry Browser	48
Start the media player with no content	49
Start the media player with content.....	49
Create a media player.....	50
Play media	53
Listen for media player events.....	54
Using rich media.....	55
Playing rich media content.....	55
Listen for rich media events.....	57
Code fragment: Managing rich media content download events.....	58
Create a custom connector for rich media connections.....	59
Code samples.....	60
Code sample: Using a raw image to recreate an encoded image.....	60
Code sample: Drawing a new bitmap using an existing bitmap.....	61
Code sample: Retrieving and displaying a rich media file	63
Code sample: Implementing a listener to download rich media content.....	64
Code sample: Implementing a custom connector	66
3 Storing data	69
Use BlackBerry persistent storage	69
BlackBerry persistent storage	69
Manage persistent data	71
Manage custom objects	73
Use the MIDP record store.....	75
Code samples.....	75
Code sample: Saving user name and password information.....	75
Code sample: Storing and viewing restaurant information	77
4 Managing data	83
Data synchronization	83
Types of data synchronization	83
Backing up and restoring data.....	84

Add support for backing up data over the wireless network	84
Access a SyncCollection	86
Notify the system when a SyncCollection changes	87
Using SyncObjects	88
Add support for backing up data with the BlackBerry Desktop Software	89
Activate synchronization when the BlackBerry device starts	90
Code samples	91
Code sample: Using a SyncCollection to back up data over the wireless network	91
Code sample: Letting the BlackBerry Desktop Software to back up and restore BlackBerry Java Application data	96
5 Implementing security and trust services	103
Connecting to an application on a SIM card	103
PIN ID restrictions for JSR177 on a BlackBerry device	103
Use JCRMI to communicate with an object on a SIM card	104
Use APDU to communicate with an object on a SIM card	106
6 Managing memory	107
Invoking garbage collection	107
Reduce the number of objects	107
Managing low memory	107
LMM triggers	107
Use the LMM	108
Free persistent objects	108
7 Creating connections	109
Fetching data using HTTP or TCP sockets	109
Working with network information	109
Determine the name of the wireless network that the BlackBerry device is registered with ...	109
Verify that the BlackBerry device is in network coverage	109
Explicitly selecting a gateway	110
Using the BlackBerry Enterprise Server as an intranet gateway	110
Using the wireless service provider's Internet gateway	110
Use HTTP connections	111
Use HTTP authentication	112
Use HTTPS connections	115
Use socket connections	115

Datagram connections.....	117
Use datagram connections.....	117
Using port connections.....	118
Use USB or serial port connections.....	119
Use Bluetooth serial port connections.....	119
Code samples.....	120
Code sample: Using an HTTP connection to retrieve data	120
Code sample: Listening for data on the serial port and rendering the data when it arrives.....	124
8 Working with Wi-Fi connections on a BlackBerry device	131
Work with wireless access families	131
Identify the wireless access families that a BlackBerry device supports.....	131
Turn on a transceiver for a wireless access family	132
Receive notifications of transceiver events	132
Work with a Wi-Fi connection	132
Query the status of the Wi-Fi transceiver on a BlackBerry device.....	132
Accessing a wireless network through a wireless access point	133
Determine if the BlackBerry device is in a wireless coverage area	134
Open a Wi-Fi connection	134
9 Creating notifications.....	135
Types of notification events	135
Add a new event source	136
Register the event source when the BlackBerry device starts	136
Triggering events	137
Respond to deferred events	138
Cancel events	139
Customize system notifications for immediate events.....	139
Code samples.....	141
Code sample: Add a new event source.....	141
Code sample: Creating a custom notification.....	144
10 Integrating with BlackBerry applications.....	147
Invoke BlackBerry applications	147
11 Managing applications.....	149
Application manager	149

Retrieve information about BlackBerry Java Applications	149
Register BlackBerry Java Applications when the BlackBerry device starts	150
Communicate with other BlackBerry Java Applications	150
Determine the services that are available to BlackBerry Java Applications	150
Listen for changes to IT policies	151
Application control	151
Allow a BlackBerry Java Application to request access to resources	151
Managing code modules	151
Retrieve module information	152
Retrieve an array of handles for existing modules on a BlackBerry device	152
Create code modules	152
The runtime store	153
Share runtime objects	153
Code sample	154
Code sample: Listening for changes to IT policies	154
12 Using the messages application	157
Create new messages	157
Work with a message	159
Open a message	161
Send a message	161
Reply to a message	162
Forward a message	163
Work with folders	163
Working with attachments	165
Create an attachment handler	165
Retrieve attachments	165
Send a message with an attachment	166
13 Using PIM applications	167
Using the calendar	167
Start the calendar from your BlackBerry Java Application	167
Use the calendar	168
Using the address book	172
Open the address book from your BlackBerry Java Application	172
Use contacts	172
Using tasks	178

Start the task application from your BlackBerry Java Application	178
Use tasks.....	179
Code samples.....	182
Code sample: Creating new recurring appointments.....	182
Code sample: Displaying a screen that lets BlackBerry device users add new contacts.....	184
Code sample: Using tasks.....	186
14 Using the phone application	189
Start the phone application from your BlackBerry Java Application.....	189
Use phone call functionality.....	189
Add DTMF tones to the send queue	190
Listen for phone events.....	191
Access and use call logs.....	191
Code sample	192
Code sample: Calculating the time spent on the phone by a participant.....	192
15 Using the BlackBerry Browser	195
Display content in the BlackBerry Browser	195
Display content in a BlackBerry Browser field	195
Code sample	198
Code sample: Using the BlackBerry Browser.....	198
16 Creating push BlackBerry Java Applications.....	203
Types of push BlackBerry Java Applications.....	203
Types of push requests.....	204
Write a client push BlackBerry Java Application.....	205
Write a server-side push application	206
Work with a server-side push request.....	208
Create a RIM push request.....	209
Create a PAP push request.....	210
Code samples.....	211
Code sample: Listening for data from a web server.....	211
Code sample: Pushing data to a BlackBerry Java Application that listens on a BlackBerry device	214
17 Localizing BlackBerry Java Applications.....	223
Storing text strings in resource files.....	223

Storing resources for a locale	223
Files required for localization.....	224
Add localization support.....	224
Retrieve strings from a resource file	226
Manage resource files for BlackBerry Java Application suites	227
Code samples.....	228
Code sample: Storing text strings in separate resources for locales.....	228
Code sample: Retrieving strings from a resource file	230
18 Testing BlackBerry Java Applications.....	233
Testing BlackBerry Java Applications using the BlackBerry IDE.....	233
Use the BlackBerry Device Simulator to test synchronizing data with the BlackBerry Desktop Software	233
Testing BlackBerry Java Applications using BlackBerry devices	234
Connect the BlackBerry IDE to a BlackBerry device	234
Debugging BlackBerry Java Applications	234
Use breakpoints.....	234
Debug a BlackBerry Java Application in the BlackBerry IDE.....	235
Manage a debugging session	236
Locate an error in the source code.....	236
Run a BlackBerry Java Application to the insertion point.....	236
Debug a BlackBerry Java Application on a BlackBerry device	236
Step through lines of code in a BlackBerry Java Application	238
View statistics to locate memory leaks	238
Display objects in memory to locate object leaks	239
View local variables.....	240
View variable or expression information.....	240
View static data	240
Evaluate (watch) Java expressions.....	241
View threads.....	241
View the data members of a process.....	242
View the call stack.....	242
View event logs	242
View classes.....	243
Optimize source code using the BlackBerry Integrated Development Environment profiler tool ..	243
Analyze code coverage	245

Start the BlackBerry email simulator.....	247
Working with compiled BlackBerry Java Applications	248
Load and remove BlackBerry Java Applications	248
View BlackBerry Java Application information	249
19 Packaging and distributing BlackBerry Java Applications	251
Preverify BlackBerry Java Applications	251
Determine if your code requires signatures	251
Controlled APIs	251
Register to use RIM controlled APIs	252
Restricted access to code signatures	253
Request code signatures.....	253
Request code signatures using a proxy server.....	254
Request a replacement registration key	254
View signature status.....	254
Distributing BlackBerry Java Applications over the wireless network	255
Distribute applications using wireless pull.....	255
Perform advanced BlackBerry Java Application distribution tasks	256
Modify information in a .jad file	257
Distributing BlackBerry Java Applications with the BlackBerry Desktop Software	259
Create an application loader file.....	259
Load a BlackBerry Java Application on a specific BlackBerry device.....	259
Specify optional components	260
Specify supported BlackBerry Device Software.....	261
A Appendix: The command line compiler	263
Using the command line compiler	263
B Appendix: XML control entity attributes.....	265
Using XML control entity attributes	265
C Appendix: .alx files	267
Elements in BlackBerry application .alx files.....	267
D Appendix: BlackBerry Java Application .jad files.....	271
Properties of BlackBerry Java Application .jad files	271
Accessing application attribute properties from a .jad file	272

	Let a BlackBerry device application retrieve name-value pair information from a .jad file	272
E	Appendix: The Eclipse development environment.....	273
	Use the Eclipse development environment	273
	Start the JDWP	273
	Connect to the Eclipse development environment	273
	Set the connection time	275
	Debug an application using the Eclipse development environment.....	276
	Acronym list.....	277

Creating user interfaces

- Elements of a BlackBerry device user interface
- Create a screen
- Adding UI components to a screen
- Creating custom UI components
- Adding menu items to BlackBerry Java Applications
- Arrange UI components
- Set field focus and navigation
- Listen for field focus changes
- Respond to UI events
- Listen for field property changes
- Manage foreground events
- Manage drawing areas
- Code samples

Elements of a BlackBerry device user interface

Screens

The main structure for a BlackBerry® device user interface is the `Screen` object. A BlackBerry® Java® Application may display more than one screen at a time, but only one screen in a BlackBerry Java Application is active at one time.

The user interface APIs initialize simple `Screen` objects. Once you create a screen, you can add fields and a menu to the screen and display it to the BlackBerry device user by pushing it on to the UI stack. The `menu` object has associated menu items that are runnable objects, which perform a specific task when the BlackBerry device user selects one of the items. For example, menu items may invoke the necessary code to establish a network connection, commit a data object to memory or close the BlackBerry Java Application. For more sophisticated custom BlackBerry Java Applications, you can customize the BlackBerry device user interface and implement new field types, as required. You can also add custom navigation and trackwheel behavior.

The `Screen` class does not implement disambiguation, which is required for complex input methods, such as international keyboards and the BlackBerry 7100 Series. For seamless integration of the different input methods, extend `Field` or one of its subclasses. Do not use `Screen` objects for typing text.

See the BlackBerry Developer Zone at <http://www.blackberry.com/developers> for knowledge base articles about displaying and working with screens.

Types of screens

Screen Type	Class	Description
Default	Screen	Use the <code>Screen</code> class to define a manager to lay out UI components on the screen and to define a specific type of screen using the styles that the constants on the <code>Field</code> superclass define.
Standard vertical	FullScreen	By default, a <code>FullScreen</code> class contains a single vertical field manager. Use a <code>FullScreen</code> class to provide an empty screen that you can add UI components to in a standard vertical layout. For another layout style, such as horizontal or diagonal, use a <code>Screen</code> class and add a <code>Manager</code> to it.
BlackBerry style	MainScreen	<p>The <code>MainScreen</code> class provides features that are common to standard BlackBerry® Java® Applications. Use a <code>MainScreen</code> object for the first screen of your BlackBerry Java Application to maintain consistency with other BlackBerry Java Applications. The <code>MainScreen</code> class provides the following UI components:</p> <ul style="list-style-type: none"> • default position of a screen title, with a <code>SeparatorField</code> after the title • main scrollable section contained in a <code>VerticalFieldManager</code> • default menu with a Close menu item • default close action when the BlackBerry device user clicks the Close menu item or presses the Escape key

How the BlackBerry JVM manages screens

The BlackBerry® JVM maintains `Screen` objects in a display stack, which is an ordered set of `Screen` objects. The screen at the top of the stack is the active screen that the BlackBerry device user sees. When a BlackBerry Java® Application displays a screen, it pushes the screen to the top of the stack. When a BlackBerry Java Application closes a screen, it removes the screen off the top of the stack and displays the next screen on the stack, redrawing it as necessary. Each screen can appear only once in the display stack. The BlackBerry JVM throws a runtime exception if a `Screen` that the BlackBerry Java Application pushes to the stack already exists. BlackBerry Java Applications must remove screens from the display stack when the BlackBerry device user finishes interacting with them so that the BlackBerry Java Application uses memory efficiently. Use only a few modal screens at one time, because each screen uses a separate thread.

UI components

Fields represent all UI components, which are rectangular regions that a `Manager` contains. A field's layout requirements determine the size of the field. Managers provide scrolling for the fields that they contain.

To create a specialized field component (such as a text field that contains multiple elements), create your own custom types by extending the `Field` class or one of its subclasses.

Traditional field	BlackBerry® Field
Button	<code>ButtonField</code>
Check box	<code>CheckboxField</code>
Date	<code>DateField</code>
Dialog box	<code>PopupScreen</code>
Drop-down list	<code>NumericChoiceField</code> or <code>ObjectChoiceField</code>
Radio button	<code>RadioButtonField</code>
Text	<code>RichTextField</code> , <code>BasicEditField</code> , <code>EditField</code> , <code>PasswordEditField</code> , or <code>AutoTextEditField</code> .
Text label	<code>LabelField</code>

Traditional field	BlackBerry® Field
List	ListField

Create a screen

- > Extend the `Screen` class or one of its subclasses, `FullScreen` or `MainScreen`.

Adding UI components to a screen

1. Create an instance of a UI component.

```
CheckboxField myCheckbox = new CheckboxField("First checkbox", true);
```

2. Add the UI component to your extension of a screen class.

```
mainScreen.add(myCheckbox);
```

Create UI components

To create an instance of a component, you can use more than one constructor. See the *API Reference* for more information on `Field` classes.

Task	Steps
Create a pop-up screen.	<ol style="list-style-type: none"> 1. Create an instance of a subclass of the <code>Manager</code> class. <pre>Manager manageLayout = new HorizontalFieldManager(VERTICAL_SCROLLBAR);</pre> 2. Create an instance of a <code>PopupScreen</code> using the <code>Manager</code> object. <pre>PopupScreen popUp = new PopupScreen(manageLayout);</pre>
Add a bitmap.	<ul style="list-style-type: none"> > Create an instance of a <code>BitmapField</code>. <pre>BitmapField myBitmapField = new BitmapField();</pre>
Create a button.	<ul style="list-style-type: none"> > Create an instance of a <code>ButtonField</code> using a style parameter. <pre>ButtonField mySubmitButton = new ButtonField("Submit");</pre>
Create a numeric drop-down list	<ul style="list-style-type: none"> > To create a drop-down list that contains numbers, create an instance of a <code>NumericChoiceField</code>. <pre>NumericChoiceField myNumericChoice = new NumericChoiceField("Select a number: ", 1, 20, 10);</pre>
Create a numeric drop-down list for a large range of numbers.	<ul style="list-style-type: none"> > Create an instance of a <code>GaugeField</code>.
Create an alphanumeric drop-down list.	<ul style="list-style-type: none"> > To create a drop-down list that contains objects, create an instance of an <code>ObjectChoiceField</code>, providing an object array as a parameter. <pre>String choiceItems[] = {"Option one", "Option two", "Option three"}; mainScreen.add(new ObjectChoiceField("Select an option:", choiceItems));</pre>
Create a check box.	<ul style="list-style-type: none"> > Create an instance of a <code>CheckboxField</code>. <pre>CheckboxField myCheckbox = new CheckboxField("First checkbox", true);</pre>

Task	Steps
Create a radio button.	<ol style="list-style-type: none"> 1. Create an instance of a <code>RadioButtonGroup</code>. <code>RadioButtonGroup rbGroup = new RadioButtonGroup();</code> 2. Create an instance of a <code>RadioButtonField</code> for each option you want to make available to the BlackBerry® device user. <code>RadioButtonField rbField = new RadioButtonField("First field");</code> <code>RadioButtonField rbField2 = new RadioButtonField("Second field");</code> 3. Invoke <code>RadioButtonGroup.add()</code> to add the <code>RadioButtonFields</code> to the <code>RadioButonGroup</code> and make sure the BlackBerry device user can select only one option at a time. <code>rbGroup.add(rbField);</code> <code>rbGroup.add(rbField2);</code>
Create a date field.	<p>> Create an instance of a <code>DateField</code>, providing the value returned by <code>System.currentTimeMillis()</code> as a parameter to return the current time.</p> <code>DateField dateField = new DateField("Date: ", System.currentTimeMillis(), DateField.DATE_TIME);</code>
Create a read-only field that you can format using different fonts and styles.	<p>> Create an instance of a <code>RichTextField</code>.</p> <code>RichTextField rich = new RichTextField("RichTextField");</code>
Create an editable text field that contains no default formatting but accepts filters.	<p>> Create an instance of a <code>BasicEditField</code>.</p> <code>BasicEditField bf = new BasicEditField("BasicEditField: ", "", 10, EditField.FILTER_UPPERCASE);</code>
Create an editable text field that lets BlackBerry® device users to access special characters.	<p>> Create an instance of an <code>EditField</code>.</p> <code>EditField edit = new EditField("EditField: ", "", 10, EditField.FILTER_DEFAULT);</code>
Create a password field.	<p>> Create an instance of a <code>PasswordEditField</code>.</p> <p>For example, the following instance uses a constructor that lets you provide a default initial value for the <code>PasswordEditField</code>.</p> <code>PasswordEditField pwd = new PasswordEditField("PasswordEditField: ", "");</code>
Create an AutoText edit field.	<p>> Create an instance of an <code>AutoTextEditField</code>.</p> <code>AutoTextEditField autoT = new AutoTextEditField("AutoTextEditField: ", "");</code> <p>Some filters render some <code>AutoText</code> entries ineffective. For example, <code>FILTER_LOWERCASE</code> renders an <code>AutoText</code> entry that contains capitalization ineffective.</p>
Create a field that displays a progress bar for the numbers that the BlackBerry® device user selects.	<p>> Create an instance of a <code>GaugeField</code>.</p> <code>GaugeField percentGauge = new GaugeField("Percent: ", 1, 100, 29, GaugeField.PERCENT)</code>
Create a text label.	<p>> Create an instance of a <code>LableField</code> to add a text label to a screen.</p> <code>LabelField title = new LabelField("UI Component Sample", LabelField.ELLIPSIS);</code>

Task	Steps
Create a field that lets a BlackBerry® device user select a range of items in the list.	<ol style="list-style-type: none"> 1. Create the items that you want to display in a ListField. <pre>String fieldOne = new String("Mark Guo"); String fieldTwo = new String("Amy Krul");</pre> 2. Create an instance of a ListField. <pre>ListField myList = new ListField();</pre> 3. Create an instance of a ListCallback. <pre>ListCallback myCallback = new ListCallback();</pre> 4. Set the call back of the ListField to be the ListCallback. <pre>myList.setCallback(myCallback);</pre> 5. Use the ListCallback object to add items to the ListField. <pre>myCallback.add(myList, fieldOne); myCallback.add(myList, fieldTwo);</pre> 6. Add the ListField to the MainScreen. <pre>mainScreen.add(myList);</pre>
Create a field that displays a folder or tree relationship between items (such as documents or message folders).	<p>A TreeField contains parent and child nodes.</p> <ol style="list-style-type: none"> 1. To draw a TreeField, implement the TreeFieldCallback interface. 2. Specify whether a folder is collapsible by invoking TreeField.setExpanded() on the TreeField object. <pre>String fieldOne = new String("Main folder"); ... TreeCallback myCallback = new TreeCallback(); TreeField myTree = new TreeField(myCallback, Field.FOCUSABLE); int node1 = myTree.addChildNode(0, fieldOne); int node2 = myTree.addChildNode(0, fieldTwo); int node3 = myTree.addChildNode(node2, fieldThree); int node4 = myTree.addChildNode(node3, fieldFour); ... int node10 = myTree.addChildNode(node1, fieldTen); myTree.setExpanded(node4, false); ... mainScreen.add(myTree);</pre> <p>Your implementation of TreeFieldCallback should add fields to the tree. See "Create a callback object" on page 33 for more information on callbacks.</p> <pre>private class TreeCallback implements TreeFieldCallback { public void drawTreeItem(TreeField _tree, Graphics g, int node, int y, int width, int indent) { String text = (String)_tree.getCookie(node); g.drawText(text, indent, y); } }</pre>

Creating custom UI components

To create custom fields, content menus, layout managers, and lists, use the BlackBerry® APIs.

Create a custom field

Task	Steps
Create a custom field.	<p>You can only add custom context menu items and custom layouts to a custom field.</p> <ul style="list-style-type: none"> > Extend the <code>Field</code> class, or one of its subclasses, implementing the <code>DrawStyle</code> interface to specify the characteristics of the custom field and turn on drawing styles. <pre> public class CustomButtonField extends Field implements DrawStyle { public static final int RECTANGLE = 1; public static final int TRIANGLE = 2; public static final int OCTAGON = 3; private String _label; private int _shape; private Font _font; private int _labelHeight; private int _labelWidth; } </pre>
Define the label, shape, and style of the custom button.	<ul style="list-style-type: none"> > Implement constructors to define the label, shape, and style of the custom button. <pre> public CustomButtonField(String label) { this(label, RECTANGLE, 0); } public CustomButtonField(String label, int shape) { this(label, shape, 0); } public CustomButtonField(String label, long style) { this(label, RECTANGLE, style); } public CustomButtonField(String label, int shape, long style) { super(style); _label = label; _shape = shape; _font = getFont(); _labelHeight = _font.getHeight(); _labelWidth = _font.getAdvance(_label); } </pre>

Task	Steps
Specify the arrangement of the objects in the field.	<ol style="list-style-type: none"> 1. Implement <code>layout()</code>. Arrange field data so that you perform the most complex calculations in <code>layout()</code> instead of in <code>paint()</code>. 2. Within your implementation, perform the following actions: <ul style="list-style-type: none"> • To calculate the available width and height, invoke <code>Math.min()</code> to return the smaller of the specified width and height and the preferred width and height of the field. • To set the required dimensions for the field, invoke <code>Field.setExtent(int, int)</code>. • Recalculate the pixel layout, cached fonts, and locale strings. <pre>protected void layout(int width, int height) { _font = getFont(); _labelHeight = _font.getHeight(); _labelWidth = _font.getAdvance(_label); width = Math.min(width, getPreferredWidth()); height = Math.min(height, getPreferredHeight()); setExtent(width, height); }</pre> <p>The manager of the field invokes <code>layout()</code> to determine how the field arranges its contents in the available space.</p>
Define the preferred width of a custom component.	<p>> Implement <code>getPreferredWidth()</code>, using the relative dimensions to make sure that the label does not exceed the dimensions of the component.</p> <pre>public int getPreferredWidth() { switch(_shape) { case TRIANGLE: if (_labelWidth < _labelHeight) { return _labelHeight << 2; } else { return _labelWidth << 1; } case OCTAGON: if (_labelWidth < _labelHeight) { return _labelHeight + 4; } else { return _labelWidth + 8; } case RECTANGLE: default: return _labelWidth + 8; } }</pre>

Task	Steps
Define the preferred height of a custom component.	<p>> Implement <code>getPreferredHeight()</code>, using the relative dimensions of the field label to determine the preferred height.</p> <pre>public int getPreferredHeight() { switch(_shape) { case TRIANGLE: if (_labelWidth < _labelHeight) { return _labelHeight << 1; } else { return _labelWidth; } case RECTANGLE: return _labelHeight + 4; case OCTAGON: return getPreferredWidth(); } return 0; }</pre>

Task	Steps
Define the appearance of the custom field.	<ol style="list-style-type: none"> 1. Perform complex calculations in <code>layout()</code> instead of in <code>paint()</code>. 2. Implement <code>paint()</code>. <pre> protected void paint(Graphics graphics) { int textX, textY, textWidth; int w = getWidth(); switch(_shape) { case TRIANGLE: int h = (w>>1); int m = (w>>1)-1; graphics.drawLine(0, h-1, m, 0); graphics.drawLine(m, 0, w-1, h-1); graphics.drawLine(0, h-1, w-1, h-1); textWidth = Math.min(_labelWidth,h); textX = (w - textWidth) >> 1; textY = h >> 1; break; case OCTAGON: int x = 5*w/17; int x2 = w-x-1; int x3 = w-1; graphics.drawLine(0, x, 0, x2); graphics.drawLine(x3, x, x3, x2); graphics.drawLine(x, 0, x2, 0); graphics.drawLine(x, x3, x2, x3); graphics.drawLine(0, x, x, 0); graphics.drawLine(0, x2, x, x3); graphics.drawLine(x2, 0, x3, x); graphics.drawLine(x2, x3, x3, x2); textWidth = Math.min(_labelWidth, w - 6); textX = (w-textWidth) >> 1; textY = (w-_labelHeight) >> 1; break; case RECTANGLE: default: graphics.drawRect(0, 0, w, getHeight()); textX = 4; textY = 2; textWidth = w - 6; break; } graphics.drawText(_label, textX, textY, (int)(getStyle() & DrawStyle.ELLIPSIS DrawStyle.HALIGN_MASK), textWidth); } </pre> <p>The fields manager invokes <code>paint()</code> to redraw the field whenever an area of the field is marked as invalid.</p>
Paint a field only within the visible region.	> Invoke <code>Graphics.getClippingRect()</code> .
Manage focus events.	> Use the <code>Field.FOCUSABLE</code> style and implement <code>Field.moveFocus()</code> .
Change the appearance of the default focus indicator.	> Override <code>Field.drawFocus()</code> .

Task	Steps
Add component capabilities.	<pre>> Implement the Field set() and get() methods. public String getLabel() { return _label; } public int getShape() { return _shape; } public void setLabel(String label) { _label = label; _labelWidth = _font.getAdvance(_label); updateLayout(); } public void setShape(int shape) { _shape = shape; updateLayout(); }</pre>

See “Code sample: Creating custom buttons” on page 35 for more information.

Create custom context menus

Task	Steps
Create the custom context menu items.	<p>> In your field class, create the custom context menu items.</p> <pre>private MenuItem myContextMenuItemA = new MenuItem(_resources, MENUITEM_ONE, 200000, 10) { public void run() { onMyMenuItemA(); } }; private MenuItem myContextMenuItemB = new MenuItem(_resources, MENUITEM_ONE, 200000, 10) { public void run() { onMyMenuItemB(); } };</pre>
Provide a context menu.	<p>> In your main BlackBerry® MDS Java Application class, override makeContextMenu().</p> <pre>protected void makeContextMenu(ContextMenu contextMenu) { contextMenu.addItem(myContextMenuItemA); contextMenu.addItem(myContextMenuItemB); }</pre>
Create the BlackBerry® MDS Java Application menu.	<p>> In your main BlackBerry Java Application class, override Screen.makeMenu(), invoking Screen.getLeafFieldWithFocus() and Field.getContextMenu() on the return value to determine which fields receive custom menu items.</p> <pre>protected void makeMenu(Menu menu) { Field focus = UiApplication.getUiApplication().getActiveScreen().getLeafFieldWithFocus(); if (focus != null) { ContextMenu contextMenu = focus.getContextMenu(); if (!contextMenu.isEmpty()) { menu.add(contextMenu); menu.addSeparator(); } } }</pre>

See “Code sample: Creating a custom context menu” on page 38 for more information.

Create custom layout managers

Task	Steps
Create a custom layout manager.	<ul style="list-style-type: none"> > Extend the <code>Manager</code> class or one of its subclasses. <pre>class DiagonalManager extends Manager { public DiagonalManager(long style){ super(style); } ... }</pre>
Return a preferred field width.	<ul style="list-style-type: none"> > Override <code>getPreferredWidth()</code> so that it returns the preferred field width for the manager. <pre>public int getPreferredWidth() { int width = 0; int numberOfFields = getFieldCount(); for (int i=0; i<numberOfFields; ++i) { width += getField(i).getPreferredWidth(); } return width; }</pre>
Organize more than one <code>TextField</code> or <code>Manager</code> object.	<ul style="list-style-type: none"> > Override the respective <code>getPreferredWidth()</code> methods for the <code>TextField</code> or <code>Manager</code> objects.
Organize multiple <code>TextFields</code> horizontally.	<ul style="list-style-type: none"> > Override <code>layout()</code>.
Return a preferred field height.	<ul style="list-style-type: none"> > Override <code>getPreferredHeight()</code> so that it returns the preferred field height for the manager. <pre>public int getPreferredHeight() { int height = 0; int numberOfFields = getFieldCount(); for (int i=0; i<numberOfFields; ++i) { height += getField(i).getPreferredHeight(); } return height; }</pre>

Task	Steps
Specify the arrangement of the child fields.	<ol style="list-style-type: none"> 1. Override <code>sublayout()</code> to retrieve the total number of fields in the manager. 2. Control how each child field is added to the screen by calling <code>setPositionChild()</code> and <code>layoutChild()</code> for each field that the manager contains. <pre>protected void sublayout(int width, int height) { int x = 0; int y = 0; Field field; int numberOfFields = getFieldCount(); for (int i=0; i<numberOfFields; ++i) { field = getField(i); layoutChild(field, width, height); setPositionChild(field, x, y); field.setPosition(x,y); x += field.getPreferredWidth(); y += field.getPreferredHeight(); } setExtent(width,height); }</pre>
Set the size of the child fields.	> In <code>sublayout()</code> , invoke <code>setExtent()</code> .
Specify how the fields receive focus.	> Override <code>nextFocus()</code> . <pre>protected int nextFocus(int direction, boolean alt) { int index = this.getFieldWithFocusIndex(); if(alt) { if(direction < 0) { // action to perform if trackwheel is rolled up } else { // action to perform if trackwheel is rolled down } } if (index == this.getFieldWithFocusIndex()) { return super.nextFocus(direction, alt); } else { return index; } }</pre>
Repaint the fields when the visible region changes.	By default, the custom manager invokes <code>paint()</code> to repaint all of the fields without regard to the clipping region. > Implement <code>subpaint()</code> .

See “Code sample: Creating a custom layout manager” on page 40 for more information.

Create custom lists

Task	Steps
Let users select multiple items in a list. Create a callback object.	<p>> Declare lists as <code>MULTI_SELECT</code>.</p> <p>> Implement the <code>ListFieldCallback</code> interface.</p> <pre>private class ListCallback implements ListFieldCallback { // The listElements vector contain the entries in the list. private Vector listElements = new Vector(); ... }</pre>
Let the field repaint a row.	<p>> Implement <code>ListFieldCallback.drawListRow()</code>, invoking <code>Graphics.drawText()</code> using parameters that specify the row to paint.</p> <pre>public void drawListRow(ListField list, Graphics g, int index, int y, int w) { String text = (String)listElements.elementAt(index); g.drawText(text, 0, y, 0, w); }</pre>
Let the field retrieve an entry from the list.	<p>> Implement <code>ListFieldCallback.get()</code>.</p> <pre>public Object get(ListField list, int index) { return listElements.elementAt(index); }</pre>
Return a preferred width for the list.	<p>> In the implementation of <code>getPreferredWidth()</code>, return a preferred width for the list.</p> <pre>public int getPreferredWidth(ListField list) { return Graphics.getScreenWidth(); }</pre>
Assign the callback and add entries to the list.	<ol style="list-style-type: none"> 1. Create the <code>ListField</code> and <code>ListCallback</code> objects for the list. <pre>ListField myList = new ListField(); ListCallback myCallback = new ListCallback();</pre> 2. Invoke <code>ListField.setCallback()</code> to associate the <code>ListFieldCallback</code> with the <code>ListField</code>. This association lets the callback add items to the list. <pre>myList.setCallback(myCallback);</pre> 3. To add entries to the list, create the entries, specify an index at which to insert each entry on the <code>ListField</code> object, and then insert each <code>ListField</code> object into the <code>ListFieldCallback</code>. <pre>String fieldOne = new String("Field one label"); String fieldTwo = new String("Field two label"); String fieldThree = new String("Field three label"); myList.insert(0); myList.insert(1); myList.insert(2); myCallback.insert(fieldOne, 0); myCallback.insert(fieldTwo, 1); myCallback.insert(fieldThree, 2); mainScreen.add(myList);</pre>

See "Code sample: Creating a custom list" on page 41 for more information.

Adding menu items to BlackBerry Java Applications

The Application Menu Item API, in the `net.rim.blackberry.api.menuitem` package, lets you add menu items to BlackBerry® Java® Applications. The `ApplicationMenuItemRepository` class lets you add or remove menu items from BlackBerry Java Applications.

Create a menu item

Task	Steps
Define a menu item.	<ul style="list-style-type: none"> > Extend the abstract <code>ApplicationMenuItem</code> class. <pre>public class SampleMenuItem extends ApplicationMenuItem { ... }</pre>
Specify the position of the menu item in the menu.	<p>A higher number means that the menu item appears lower in the menu.</p> <ul style="list-style-type: none"> > Invoke <code>ApplicationMenuItem()</code> <pre>SampleMenuItem() { super(20); }</pre>
Specify the menu item text.	<ul style="list-style-type: none"> > Implement <code>toString()</code>. <pre>public String toString() { return "Open the Contacts Demo application"; }</pre>
Specify the behaviour of the menu item.	<ul style="list-style-type: none"> > Implement <code>run()</code>. <pre>public Object run(Object context) { Contact c = (Contact)context; // An error occurs if this does not work. if (c != null) { new ContactsDemo().enterEventDispatcher(); } else { throw new IllegalStateException("Context is null, expected a Contact instance"); } Dialog.alert("Viewing a message in the messaging view"); return null; }</pre>

Register a menu item

Task	Steps
Retrieve the BlackBerry® MDS Java Application menu item repository.	<pre>> Invoke ApplicationMenuItemRepository.getInstance(). ApplicationMenuItemRepository repository = ApplicationMenuItemRepository.getInstance();</pre>
Create your BlackBerry Java Application menu item.	<pre>> Invoke the constructor. TestApplicationMenuItem tami = new TestApplicationMenuItem();</pre>
Add the menu item to the repository.	<pre>> Invoke ApplicationMenuItemRepository.addItem(). repository.addItem(ApplicationMenuItemRepository.MENUITEM_ADDRESSCARD _VIEW, tami);</pre>
Add the menu item to BlackBerry® Maps.	<pre>> Invoke ApplicationMenuItemRepository.addItem() using the MENUITEM_MAPS field. repository.addItem(ApplicationMenuItemRepository.MENUITEM_MAPS, tami);</pre>

See "Code sample: Creating a new menu item in a BlackBerry Java Application" on page 42 for more information.

Arrange UI components

To arrange components on a screen, use BlackBerry® API layout managers .

The following four classes extend the `Manager` class to provide predefined layout managers:

- `VerticalFieldManager`
- `HorizontalFieldManager`
- `FlowFieldManager`
- `DialogFieldManager`

To create a custom layout manager, extend `Manager`.

Define a layout manager

Task	Steps
Create a layout manager.	<p>On an instance of a <code>Screen</code>, complete the following actions:</p> <ol style="list-style-type: none"> 1. Instantiate the appropriate <code>Manager</code> subclass. 2. Add UI components to the layout manager. 3. Add the layout manager to the screen. <pre>VerticalFieldManager vfm = new VerticalFieldManager(Manager.VERTICAL_SCROLL); vfm.add(bitmapField); vfm.add(bitmapField2); ... mainScreen.add(vfm)</pre>

Set field focus and navigation

`UIEventListeners` let BlackBerry® Java® Applications respond to a change to a UI object.

Listen for field focus changes

1. Implement `FocusChangeListener`. Your implementation of `FocusChangeListener` should specify what action occurs when the field gains, loses, or changes the focus by implementing `focusChanged()`.
2. Assign your implementation to a `Field` by invoking `setChangeListener()`.

```
private class FocusListener implements FocusChangeListener {
    public void focusChanged(Field field, int eventType) {
        if (eventType == FOCUS_GAINED) {
            // Perform action when this field gains the focus.
        }
        if (eventType == FOCUS_CHANGED) {
            // Perform action when the focus changes for this field.
        }
        if (eventType == FOCUS_LOST) {
            // Perform action when this field loses focus.
        }
    }
}
FocusListener myFocusChangeListener = new FocusListener();
myField.setFocusChangeListener(myFocusChangeListener);
```

Respond to UI events

Task	Steps
Respond to UI navigation events.	<ul style="list-style-type: none"> > Manage navigation events by extending the <code>net.rim.device.api.ui.Screen</code> class (or one of its subclasses) and overriding the following navigation methods: <ul style="list-style-type: none"> • <code>navigationClick(int status, int time)</code> • <code>navigationUnclick(int status, int time)</code> • <code>navigationMovement(int dx, int dy, int status, int time)</code> • When you create a new UI BlackBerry® Java® Application, use the new Screen navigation methods and avoid using the <code>TrackwheelListener</code>. • If your existing UI BlackBerry Java Application implements the <code>TrackwheelListener</code>, update your BlackBerry Java Application to use the new Screen navigation methods.
Interpret the status parameter of the navigation methods.	<ul style="list-style-type: none"> > In your implementation of one of the <code>navigationClick</code>, <code>navigationUnclick</code>, or <code>navigationMovement</code> methods of the <code>Screen</code> or <code>Field</code> classes, perform a bitwise AND operation on the status parameter to yield more information about the event. For example, to determine the type of input mechanism that triggered an event, in your implementation of the <code>navigationClick(int status, int time)</code> method, create code such as the following: <pre>public boolean navigationClick(int status, int time) { if ((status & KeypadListener.STATUS_TRACKWHEEL) == KeypadListener.STATUS_TRACKWHEEL) { //Input came from the trackwheel } else if ((status & KeypadListener.STATUS_FOUR_WAY) == KeypadListener.STATUS_FOUR_WAY) { //Input came from a four way navigation input device } return super.navigationClick(status, time); } </pre> <p>See the <i>API Reference</i> for the class <code>net.rim.device.api.system.KeypadListener</code> for a listing of other status modifiers.</p>

Task	Steps
Respond to BlackBerry® device user interaction.	<ul style="list-style-type: none"> > Use the <code>Screen</code> class and its subclasses to provide a menu for the BlackBerry device users to perform actions.
Provide screen navigation when using a <code>FullScreen</code> or <code>Screen</code> .	<p>Creating a <code>MainScreen</code> object provides default navigation to your BlackBerry® Java® Application. Avoid using buttons or other UI elements that take up space on the screen.</p> <ul style="list-style-type: none"> > Specify the <code>DEFAULT_MENU</code> and <code>DEFAULT_CLOSE</code> parameters in the constructor to provide default navigation. <pre>FullScreen fullScreen = new FullScreen(DEFAULT_MENU DEFAULT_CLOSE);</pre>
Provide menu support.	<ul style="list-style-type: none"> > Extend the <code>Screen</code> class.
Provide menu support in a BlackBerry® Java® Application that uses the <code>TrackwheelClick()</code> method of the <code>TrackwheelListener</code> .	<ol style="list-style-type: none"> 1. Update your BlackBerry Java Application to use an extension of the <code>Screen</code> class. 2. In the constructor of your <code>Screen</code> class extension, make sure to invoke the <code>Screen</code> class constructor using the <code>DEFAULT_MENU</code> property. 3. Make sure your extension of the <code>makeMenu()</code> method of the <code>Screen</code> class invokes <code>Screen.makeMenu()</code> and adds the required menu items for the current UI BlackBerry Java Application.

Task	Steps
Manage selected menu items.	<p>Perform the actions in one of the following options:</p> <p>Option 1</p> <ol style="list-style-type: none"> 1. Override the <code>onMenu()</code> method. 2. In your extension of <code>makeMenu()</code> cache a reference to the "menu" parameter in the screen. 3. In your extension of <code>OnMenu()</code>, invoke <code>Screen.OnMenu()</code>. 4. In your code, inspect the cached <code>Menu</code> object to determine which menu item the BlackBerry® device user selected. 5. Use the result of this inspection to trigger the appropriate menu action. <p>Option 2</p> <ol style="list-style-type: none"> 1. Extend the <code>MenuItem</code> class. <pre>private MenuItem viewItem = new MenuItem("View Message", 100, 10);</pre> 2. Create a <code>run()</code> method that implements the behavior that you expect to occur when the BlackBerry device user clicks a menu item. When a BlackBerry device user selects a <code>MenuItem</code>, this action invokes the <code>run()</code> method. <pre>public void run() { Dialog.inform("This is today's message"); }</pre> 3. If you do not use localization resources, override <code>toString()</code> to specify the name of the menu item. 4. When you add your own menu items, define a <code>Close</code> menu item explicitly. <pre>private MenuItem closeItem = new MenuItem("Close", 200000, 10) { public void run() { onClose(); } }</pre> 5. To add the menu items to the screen, override <code>Screen.makeMenu()</code>, adding instances of the extended <code>MenuItem</code> class. <pre>protected void makeMenu(Menu menu, int instance) { menu.add(viewItem); menu.add(closeItem); }</pre> 6. In your extension of the <code>MenuItem</code> class, do not override the <code>onMenu()</code> method.

Listen for field property changes

1. Implement the `FieldChangeListener` interface.
2. Assign your implementation to a field by invoking `setChangeListener()`.

```
private class FieldListener implements FieldChangeListener {
    public void fieldChanged(Field field, int context) {
        if (context != FieldChangeListener.PROGRAMMATIC) {
            // Perform action if user changed field.
        } else {
            // Perform action if application changed field.
        }
    }
}
// ...
```

```
FieldListener myFieldChangeListener = new FieldListener()
myField.setChangeListener(myFieldChangeListener);
```

Manage foreground events

The system calls `Application.activate()` when it brings a BlackBerry® Java® Application to the foreground.

Manage drawing areas

The `Graphics` object represents the entire drawing surface that is available to the BlackBerry® Java® Application. To limit this area, divide it into `XYRect` objects. Each `XYPoint` represents a point on the screen, which is composed of an X co-ordinate and a Y co-ordinate.

Task	Steps
Create rectangular clipping areas.	<ol style="list-style-type: none"> 1. Create an instance of an <code>XYPoint</code> object and an <code>XYRect</code> object. <pre>XYPoint bottomRight = new XYPoint(50, 50); XYRect rectangle = new XYRect(topLeft, bottomRight); XYPoint topLeft = new XYPoint(10, 10);</pre> 2. Invoke <code>pushContext()</code> or <code>pushRegion()</code>. 3. When you make drawing calls with <code>pushContext()</code>, specify that the region origin should not adjust the drawing offset. <pre>graphics.pushContext(rectangle, 0, 0); graphics.fillRect(10, 10, 30, 30); graphics.drawRect(15, 15, 30, 30); graphics.popContext();</pre> 4. When you invoke drawing methods by first calling <code>pushRegion()</code>, specify that the region origin should adjust the drawing offset. <pre>graphics.pushRegion(rectangle); graphics.fillRect(10, 10, 30, 30); graphics.drawRect(15, 15, 30, 30); graphics.popContext();</pre>
Invert an area.	<ol style="list-style-type: none"> 1. Invert a specified <code>XYRect</code> object. 2. Specify the portion of the <code>Graphics</code> object to push onto the stack. 3. After you invoke <code>pushContext()</code> (or <code>pushRegion()</code>), provide the portion of the <code>Graphics</code> object to invert. <pre>graphics.pushContext(rectangle); graphics.invert(rectangle); // invert the entire XYRect object graphics.popContext();</pre>
Translate an area.	<ol style="list-style-type: none"> > Invoke <code>translate()</code>. The <code>XYRect</code> is translated from its origin of (1,1) to an origin of (20,20). After translation, the bottom portion of the <code>XYRect</code> object extends past the bounds of the <code>graphics</code> context and clips it. <pre>XYRect rectangle = new XYRect(1, 1, 100, 100); XYPoint newLocation = new XYPoint(20, 20); rectangle.translate(newLocation);</pre>

Code samples

Code sample: Creating custom buttons

Example: CustomButtonField.java

```
/**
 * CustomButtonField.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.custombuttons;

import net.rim.device.api.ui.*;
import net.rim.device.api.system.*;

/**
 * CustomButtonField is a class that creates button fields of various
 * shapes. This sample demonstrates how to create custom UI fields.
 */

public class CustomButtonField extends Field implements DrawStyle {
    public static final int RECTANGLE = 1;
    public static final int TRIANGLE = 2;
    public static final int OCTAGON = 3;

    private String _label;
    private int _shape;
    private Font _font;
    private int _labelHeight;
    private int _labelWidth;

    /* Constructs a button with specified label, and the default style and shape. */
    public CustomButtonField(String label) {
        this(label, RECTANGLE, 0);
    }

    /* Constructs a button with specified label and shape, and the default style. */
    public CustomButtonField(String label, int shape) {
        this(label, shape, 0);
    }

    /* Constructs a button with specified label and style, and the default shape. */
    public CustomButtonField(String label, long style) {
        this(label, RECTANGLE, style);
    }

    /* Constructs a button with specified label, shape, and style */
    public CustomButtonField(String label, int shape, long style) {
        super(style);
        _label = label;
        _shape = shape;
        _font = getFont();
        _labelHeight = _font.getHeight();
        _labelWidth = _font.getAdvance(_label);
    }
}
```

```

/* Method that draws the focus indicator for this button and
 * inverts the inside region of the shape.
 */
protected void drawFocus(Graphics graphics, boolean on) {
    switch(_shape) {
        case TRIANGLE:
            int w = getWidth();
            int h = w >> 1;
            for (int i=h-1; i>=2; --i) {
                graphics.invert(i, h - i, w - (i << 1), 1);
            }
            break;
        case RECTANGLE:
            graphics.invert(1, 1, getWidth() - 2, getHeight() - 2);
            break;
        case OCTAGON:
            int x3 = getWidth();
            int x = 5 * x3 / 17;
            int x2 = x3 - x;
            x3 = x3 - 1;
            x2 = x2 - 1;
            graphics.invert(1, x, getWidth() - 2, x2 - x + 1);

            for (int i=1; i<x; ++i) {
                graphics.invert(1+i, x-i,
                    getWidth() - ((i+1)<<1), 1);
                graphics.invert(1+i, x2+i,
                    getWidth() - ((i+1)<<1), 1);
            }
            break;
    }
}

/* Returns the label. */
public String getLabel() {
    return _label;
}

/* Returns the shape. */
public int getShape() {
    return _shape;
}

/* Sets the label. */
public void setLabel(String label) {
    _label = label;
    _labelWidth = _font.getAdvance(_label);

    updateLayout();
}

/* Sets the shape. */
public void setShape(int shape) {
    _shape = shape;
    updateLayout();
}

/* Retrieves the preferred width of the button. */
public int getPreferredWidth() {

```

```

        switch(_shape) {
            case TRIANGLE:
                if (_labelWidth < _labelHeight) {
                    return _labelHeight << 2;
                } else {
                    return _labelWidth << 1;
                }
            case OCTAGON:
                if (_labelWidth < _labelHeight) {
                    return _labelHeight + 4;
                } else {
                    return _labelWidth + 8;
                }
            case RECTANGLE: default:
                return _labelWidth + 8;
        }
    }

    /* Retrieves the preferred height of the button. */
    public int getPreferredHeight() {
        switch(_shape) {
            case TRIANGLE:
                if (_labelWidth < _labelHeight) {
                    return _labelHeight << 1;
                } else {
                    return _labelWidth;
                }
            case RECTANGLE:
                return _labelHeight + 4;
            case OCTAGON:
                return getPreferredWidth();
        }
        return 0;
    }

    /* Lays out this button's contents.
     * This field's manager invokes this method during the layout
     * process to instruct this field to arrange its contents, given an
     * amount of available space.
     */
    protected void layout(int width, int height) {
        // Update the cached font in case it has been changed.
        _font = getFont();
        _labelHeight = _font.getHeight();
        _labelWidth = _font.getAdvance(_label);

        // Calculate width.
        width = Math.min( width, getPreferredWidth() );
        // Calculate height.
        height = Math.min( height, getPreferredHeight() );
        // Set dimensions.
        setExtent( width, height );
    }

    /*
     * Redraws this button. The field's manager invokes this method during the
     * repainting process to instruct this field to repaint itself.
     */
    protected void paint(Graphics graphics) {

```

```

int textX, textY, textWidth;
int w = getWidth();
switch(_shape) {
    case TRIANGLE:
        int h = (w>>1);
        int m = (w>>1)-1;
        graphics.drawLine(0, h-1, m, 0);
        graphics.drawLine(m, 0, w-1, h-1);
        graphics.drawLine(0, h-1, w-1, h-1);

        textWidth = Math.min(_labelWidth,h);
        textX = (w - textWidth) >> 1;
        textY = h >> 1;
        break;
    case OCTAGON:
        int x = 5*w/17;
        int x2 = w-x-1;
        int x3 = w-1;
        graphics.drawLine(0, x, 0, x2);
        graphics.drawLine(x3, x, x3, x2);
        graphics.drawLine(x, 0, x2, 0);
        graphics.drawLine(x, x3, x2, x3);
        graphics.drawLine(0, x, x, 0);
        graphics.drawLine(0, x2, x, x3);
        graphics.drawLine(x2, 0, x3, x);
        graphics.drawLine(x2, x3, x3, x2);
        textWidth = Math.min(_labelWidth, w - 6);
        textX = (w-textWidth) >> 1;
        textY = (w-_labelHeight) >> 1;
        break;
    case RECTANGLE: default:
        graphics.drawRect(0, 0, w, getHeight());
        textX = 4;
        textY = 2;
        textWidth = w - 6;
        break;
}
graphics.drawText(_label, textX, textY, (int)(getStyle() &
    DrawStyle.ELLIPSIS | DrawStyle.HALIGN_MASK ),
    textWidth );
}
}

```

Code sample: Creating a custom context menu

Example: ContextMenuSample.java

```

/**
 * ContextMenuSample.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.contextmenus;

```

```

import net.rim.device.api.i18n.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import com.rim.samples.docs.resource.*;

public class ContextMenuSample extends UiApplication implements ContextMenuSampleResource
{
    private MyContextField myContextField;

    private static ResourceBundle _resources = ResourceBundle.getBundle(
        ContextMenuSampleResource.BUNDLE_ID,
        ContextMenuSampleResource.BUNDLE_NAME);

    public static void main(String[] args) {
        ContextMenuSample app = new ContextMenuSample();
        app.enterEventDispatcher();
    }

    // Inner class to define a new field.
    private static class MyContextField extends RichTextField {
        private MenuItem myContextMenuItemA = new MenuItem(
            _resources, MENUITEM_ONE, 200000, 10) {
            public void run() {
                onMyMenuItemA();
            }
        };
        private MenuItem myContextMenuItemB = new MenuItem(
            _resources, MENUITEM_TWO, 200000, 10) {
            public void run() {
                onMyMenuItemB();
            }
        };

        private void onMyMenuItemA() {
            // Perform an action when user selects menu item.
        }
        private void onMyMenuItemB() {
            // Perform an action when user selects menu item.
        }

        protected void makeContextMenu(ContextMenu contextMenu) {
            contextMenu.addItem(myContextMenuItemA);
            contextMenu.addItem(myContextMenuItemB);
        }

        MyContextField(String text) {
            super(text);
        }
    }

    public ContextMenuSample() {
        MainScreen mainScreen = new MainScreen();

        MyContextField myContextField = new MyContextField("Field label: ");
        mainScreen.add(myContextField);
    }
}

```

```

        pushScreen(mainScreen);
    }
}

```

Code sample: Creating a custom layout manager

Example: DiagonalManager.java

```

/**
 * DiagonalManager.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.custommenu;

import net.rim.device.api.system.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;

class DiagonalManager extends Manager {
    public DiagonalManager(long style) {
        super(style);
    }

    public int getPreferredWidth() {
        int width = 0;
        int numberOfFields = getFieldCount();
        for (int i=0; i<numberOfFields; ++i) {
            width += getField(i).getPreferredWidth();
        }
        return width;
    }

    public int getPreferredHeight() {
        int height = 0;
        int numberOfFields = getFieldCount();
        for (int i=0; i<numberOfFields; ++i) {
            height += getField(i).getPreferredHeight();
        }
        return height;
    }

    protected void sublayout(int width, int height) {
        int x = 0;
        int y = 0;
        Field field;
        int numberOfFields = getFieldCount();
        for (int i=0; i<numberOfFields; ++i) {
            field = getField(i);
            layoutChild( field, width, height );
        }
    }
}

```

```

        setPositionChild(field, x, y);
        x += field.getPreferredWidth();
        y += field.getPreferredHeight();
    }
    setExtent(width,height);
}

protected int nextFocus(int direction, boolean alt) {
    int index = this.getFieldWithFocusIndex();
    if(alt) {
        if(direction < 0) {
            // action to perform if trackwheel is rolled up
        } else {
            // action to perform if trackwheel is rolled down
        }
    }
    if (index == this.getFieldWithFocusIndex()) {
        return super.nextFocus(direction, alt);
    } else {
        return index;
    }
}
}

```

Code sample: Creating a custom list

Example: SampleListFieldCallback.java

```

/**
 * SampleListFieldCallback.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.listfields;

import java.util.*;
import net.rim.device.api.system.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;

public class SampleListFieldCallback extends UiApplication {
    private ListField myList;
    public static void main(String[] args) {
        SampleListFieldCallback app = new SampleListFieldCallback();
        app.enterEventDispatcher();
    }
    private static class ListCallback implements ListFieldCallback {
        private Vector listElements = new Vector();
        public void drawListRow(
            ListField list, Graphics g, int index, int y, int w) {
            String text = (String)listElements.elementAt(index);
            g.drawText(text, 0, y, 0, w);
        }
    }
}

```

```

        public Object get(ListField list, int index) {
            return listElements.elementAt(index);
        }
        public int indexOfList(ListField list, String p, int s) {
            return listElements.indexOf(p, s);
        }
        public int getPreferredWidth(ListField list) {
            return Graphics.getScreenWidth();
        }
        public void insert(String toInsert, int index) {
            listElements.addElement(toInsert);
        }
        public void erase() {
            listElements.removeAllElements();
        }
    }
    public SampleListFieldCallback() {
        MainScreen mainScreen = new MainScreen();
        myList = new ListField();
        ListCallback myCallback = new ListCallback();
        myList.setCallback(myCallback);
        String fieldOne = "ListField one";
        String fieldTwo = "ListField two";
        String fieldThree = "ListField three";

        myList.insert(0);
        myCallback.insert(fieldOne, 0);
        myList.insert(1);
        myCallback.insert(fieldTwo, 1);
        myList.insert(2);
        myCallback.insert(fieldThree, 2);

        mainScreen.add(myList);

        pushScreen(mainScreen);
    }
}

```

Code sample: Creating a new menu item in a BlackBerry Java Application

The menu item appears when a BlackBerry® device user views a contact in the address book. When a BlackBerry device user clicks the menu item, the ContactsDemo application appears.

Example: DemoAppMenuItem.java

```

/**
 * DemoApplicationMenuItem.java
 * Copyright (C) 2003-2007 Research In Motion Limited.
 *
 * The following code example creates a menu item that appears when
 * a user views a contact in the address book. When a user clicks the menu item,
 * the Contacts Demo application appears.
 */

package com.rim.samples.docs.menuitem;

```

```

import net.rim.device.api.system.*;
import net.rim.device.api.ui.component.Dialog.*;
import net.rim.blackberry.api.menuitem.*;
import net.rim.blackberry.api.pdap.*;
import javax.microedition.pim.*;

public final class DemoAppMenuItem extends Application
{
    private static final String ARG_LAUNCH_CONTACT_DEMO = "1";
    //private static final String ARG_LAUNCH_APP2 = "2";
    //... etc

    public static void main(String[] args) {
        if(args == null || args.length == 0)
        {
            DemoAppMenuItem app = new DemoAppMenuItem();
            app.enterEventDispatcher();
        }
        else
        {
            String appToLaunch = args[0];
            if(ARG_LAUNCH_CONTACT_DEMO.equals(appToLaunch))
            {
                new
com.rim.samples.docs.contactsdemo.ContactsDemo().enterEventDispatcher();

            }

            //add more else ifs here
        }

        DemoAppMenuItem() {
            long locationToAddMenuItem =
ApplicationMenuItemRepository.MENUITEM_ADDRESSCARD_VIEW;
            addMenuItem(ARG_LAUNCH_CONTACT_DEMO, locationToAddMenuItem, new
ContactsDemoMenuItem());

            System.exit(0);
        }

        private static void addMenuItem(String argOfAppl, long location, ApplicationMenuItem
applMenuItem)
        {
            ApplicationMenuItemRepository amir = ApplicationMenuItemRepository.getInstance();
            ApplicationDescriptor app = ApplicationDescriptor.currentApplicationDescriptor();

            //set the argument so that we know which app we want to have launched
            app = new ApplicationDescriptor(app, new String[]{ARG_LAUNCH_CONTACT_DEMO});

            amir.addMenuItem(location, applMenuItem, app);
        }

        /**
         * Create the menu item classes here

```

```
*/
private static class ContactsDemoMenuItem extends ApplicationMenuItem {
    ContactsDemoMenuItem() {
        super(20);
    }

    public String toString() {
        return "Open the Contacts Demo";
    }

    public Object run(Object context) {
        BlackBerryContact c = (BlackBerryContact)context; //an error if this doesn't
work
        if ( c != null ) {
            Application.getApplication().requestForeground();
            //on invocation, will call the main method of this app. with argument as
specified in addMenuItem
        } else {
            throw new IllegalStateException( "Context is null, expected a Contact
instance");
        }
        return null;
    }
}
}
```

Using graphics and multimedia

Using images
Drawing and rendering images
Using audio
Using rich media
Code samples

Using images

Use raw images

Task	Steps
Allow BlackBerry® Java® Applications to use raw image data.	<p>> To retrieve raw image data from a specified region of a bitmap and store the data in an integer array, invoke <code>Bitmap.getARGB()</code>.</p> <pre>void getARGB(int[] argbData, int offset, int scanLength, int x, int y, int width, int height);</pre>
Retrieve image data.	<ol style="list-style-type: none"> 1. Initialize an integer array. 2. To store the raw image data of the new or predefined bitmap in the integer array, invoke <code>Bitmap.getARGB()</code>. <pre>Bitmap original = Bitmap.getPredefinedBitmap(Bitmap.INFORMATION); int[] argb = new int[original.getWidth() * original.getHeight()]; original.getARGB(argb, 0, original.getWidth(), 0, 0, original.getWidth(), original.getHeight());</pre>
Compare two images to see if they are identical.	<p>> Invoke <code>Bitmap.equals()</code>.</p> <pre>if(restored.equals(original)) { System.out.println("Success! Bitmap renders correctly with RGB data."); } else if(!restored.equals(original)) { System.out.println("Bitmap rendered incorrectly with RGB data."); }</pre>

Use encoded images

Task	Steps
Access an image.	<ol style="list-style-type: none"> 1. Save an image to the project folder or sub-folder. 2. Add the image to the project in the BlackBerry® Integrated Development Environment. 3. Invoke <code>Class.getResourceAsStream()</code> to retrieve the image as an input stream of bytes. <pre>private InputStream input; ... try { input = Class.forName("com.rim.samples.docs.imagedemo.ImageDemo"). getResourceAsStream("/images/example.png"); } catch (ClassNotFoundException e) { System.out.println("Class not found"); }</pre>
Encode an image.	<ol style="list-style-type: none"> 1. Invoke <code>EncodedImage.createEncodedImage()</code>. This method creates an instance of <code>EncodedImage</code> using the raw image data in the byte array. 2. Check for an <code>IllegalArgumentException</code>, which <code>EncodedImage.createEncodedImage()</code> throws if the byte array that you provide as a parameter does not contain a recognized image format. <pre>private byte[] data = new byte[2430]; // Store the contents of the image file. try { input.read(data); // Read the image data into the byte array. } catch (IOException e) { // Handle exception. } try { EncodedImage image = EncodedImage.createEncodedImage(data, 0, data.length); } catch (IllegalArgumentException iae) { System.out.println("Image format not recognized."); }</pre>
Display an encoded image.	<ol style="list-style-type: none"> 1. To assign the encoded image to a <code>BitmapField</code>, invoke <code>BitmapField.setImage()</code>. 2. To add the <code>BitmapField</code> to the screen, invoke <code>add()</code>. <pre>BitmapField field = new BitmapField(); field.setImage(image); add(field);</pre>
Set the decoding mode.	<ol style="list-style-type: none"> 1. Invoke <code>EncodedImage.setDecodeMode()</code>. 2. Provide one of the following modes as a parameter to the method: <ul style="list-style-type: none"> • <code>DECODE_ALPHA</code>: decodes an alpha channel, if one exists (this is the default mode) • <code>DECODE_NATIVE</code>: forces the BlackBerry® Java® Application to decode the bitmap to the native bitmap type of the handheld software application • <code>DECODE_READONLY</code>: marks the decoded bitmap as read-only •
Set the image display size.	<p>> Invoke <code>EncodedImage.setScale()</code>.</p> <p>The inverse of the integer specified by the scale parameter scales the image. For example, if you set the scaling factor to 2, the image decodes at 50% of its original size.</p>

See "Code sample: Using a raw image to recreate an encoded image" on page 60 for more information.

Drawing and rendering images

Position an image

Task	Steps
Use an individual field.	<ol style="list-style-type: none"> 1. Invoke the <code>Graphics()</code> constructor. <pre>Bitmap surface = new Bitmap(100, 100); BitmapField surfaceField = new BitmapField(surface); Graphics graphics = new Graphics(surface);</pre> 2. Add the <code>BitmapField</code> to the screen. <pre>mainScreen.add(surfaceField);</pre>
Use the whole screen.	<ol style="list-style-type: none"> 1. Invoke <code>Screen.getGraphics()</code>. <pre>Graphics graphics = Screen.getGraphics();</pre> 2. Make sure your methods perform their drawing functions within the boundaries of the screen. <pre>graphics.fillRect(10, 10, 30, 30); graphics.drawRect(15, 15, 30, 30);</pre>

Draw an image in color

Task	Steps
Determine whether the BlackBerry® device supports color display.	<ul style="list-style-type: none"> > Invoke <code>Graphics.isColor()</code>.
Determine the number of colors that the BlackBerry® device supports.	<ul style="list-style-type: none"> > Invoke <code>Graphics.getNumColors()</code>.
Set the pixel transparency in the drawing area.	<ol style="list-style-type: none"> 1. Invoke one of the following methods: <ul style="list-style-type: none"> • <code>Graphics.setGlobalAlpha()</code> • <code>Graphics.getGlobalAlpha()</code> 2. Define a global alpha value within the following range: <ul style="list-style-type: none"> • 0 (0x0000): completely transparent • 255 (0x00FF): fully opaque
Determine raster operations that the BlackBerry® Java® Application supports.	<ol style="list-style-type: none"> 1. Invoke <code>Graphics.isRopSupported(int)</code>. 2. Provide one of the following constants as a parameter: <ul style="list-style-type: none"> • <code>ROP_CONST_GLOBALALPHA</code>: blends the constant foreground color using a constant global alpha value with destination pixels • <code>ROP_SRC_GLOBALALPHA</code>: blends a source bitmap using a constant global alpha value with destination pixels •

Task	Steps
Draw a set of shaded, filled paths.	<pre>> Invoke Graphics.drawShadedFilledPath(): public void drawShadedFilledPath(int[] xPts, int[] yPts, byte[] pointTypes, int[] colors, int[] offsets);</pre> <p>The following example draws a path that blends from blue to red:</p> <pre>Bitmap surface = new Bitmap(240, 160); BitmapField surfaceField = new BitmapField(surface); add(surfaceField); Graphics graphics = new Graphics(surface); int[] X_PTS = { 0, 0, 240, 240 }; int[] Y_PTS = { 20, 50, 50, 20 }; int[] drawColors = { 0x0000CC, 0x0000CC, 0xCC0000, 0xCC0000 }; try { graphics.drawShadedFilledPath(X_PTS, Y_PTS, null, drawColors, null); } catch (IllegalArgumentException iae) { System.out.println("Bad arguments."); }</pre>
Turn a drawing style on.	> Invoke <code>Graphics.setDrawingStyle(int drawStyle, boolean on)</code> .
Turn a drawing style off.	> Invoke <code>Graphics.setDrawingStyle(int drawStyle, boolean off)</code> .
Determine if a drawing style is set.	> Invoke <code>Graphics.isDrawingStyleSet(int drawStyle)</code> .
Use a monochrome bitmap as a stamp.	<p>The STAMP_MONOCHROME option enables BlackBerry® Java® Applications to use monochrome bitmaps as stamps by rendering the nontransparent region in color. This option applies to bitmaps that are 1 bit and have alpha defined.</p> <pre>BitmapField field = new BitmapField(original, BitmapField.STAMP_MONOCHROME);</pre>
Draw an image on an empty bitmap.	<ol style="list-style-type: none"> 1. Create an empty bitmap. The example below copies the type and size from an existing bitmap. 2. Create a Graphics object using the empty bitmap as the drawing surface. 3. To draw a new image using raw data retrieved from the original, invoke <code>Graphics.rawRGB()</code>. <pre>Bitmap restored = new Bitmap(original.getType(), original.getWidth(), original.getHeight()); Graphics graphics = new Graphics(restored); try { graphics.drawRGB(argb, 0, restored.getWidth(), 0, 0, restored.getWidth(), restored.getHeight()); } catch (Exception e) { System.out.println("Error occurred during drawing: " + e); }</pre>

See “Code sample: Drawing a new bitmap using an existing bitmap” on page 61 for more information.

Using audio

Start the media player from the BlackBerry Browser

1. Invoke `Browser.getDefaultSession()`.

```
BrowserSession soundclip = Browser.getDefaultSession();
```

2. Invoke `BrowserSession.displayPage()`.
`soundclip.displayPage("file:///SDCard/BlackBerry/music/TarzanYell.mp3");`

Start the media player with no content

1. Import the `javax.microedition.content` package.
`Import javax.microedition.content;`
2. Invoke `Registry.getRegistry()`, storing a reference to the returned object in a `Registry` object. The *classname* parameter is the name of the class in the application that extends `javax.microedition.midlet.MIDlet`, `net.rim.device.api.system.Application` or `net.rim.device.api.ui.UiApplication`.
`Registry reg = Registry.getRegistry(String classname);`
3. Create a new instance of an `Invocation` object, storing a reference to the object in an `Invocation` object.
`Invocation invocation = new Invocation(null, null, BlackBerryInvocation.CONTENT_HANDLER_MEDIA_PLAYER);`
4. Invoke `Registry.invoke(Invocation invocation)` using the new `Invocation` object as a parameter.
`reg.invoke(invocation);`

Start the media player with content

1. Import the `javax.microedition.content` package.
`Import javax.microedition.content;`
2. Invoke `Registry.getRegistry()`, storing a reference to the returned object in a `Registry` object. The *classname* parameter is the name of the class in the application that extends `javax.microedition.midlet.MIDlet`, `net.rim.device.api.system.Application` or `net.rim.device.api.ui.UiApplication`.
`Registry reg = Registry.getRegistry(String classname);`
3. Create a new instance of an `Invocation` object, use a media type supported by the media player as a parameter, and store a reference to the object in an `Invocation` object.
`Invocation invocation = new Invocation(file://...);`
4. Invoke `Registry.invoke(Invocation invocation)` using the new `Invocation` object as a parameter.
`reg.invoke(invocation);`

Create a media player

To play audio on a BlackBerry® device, use the API items in the `javax.microedition.media` package (JSR 135) to create a media player and then add functionality to it.

Task	Steps
Create a player for a sequence of tones.	<ul style="list-style-type: none"> > Use the <code>ToneControl</code> to permit playback of a BlackBerry® device user-defined sequence of tones in an unvarying pitch. See "Access media player functionality" on page 89 for more information on media player controls. <p>Tempo is the beats per minute with 1 beat equal to 1/4 note. You determine the tempo by multiplying the tempo modifier by 4 to keep it within the byte range of 1 to 127. Tempos in the range of 20 bpm to 508 bpm equate to a tempo modifier range of 5 to 127.</p>
Create a player for media from a URL.	<ul style="list-style-type: none"> > Invoke <code>Manager.createPlayer(String locator)</code>. The string parameter must use URI syntax that describes the media content.
Create a player for media from an input stream.	<ol style="list-style-type: none"> 1. Invoke <code>Manager.createPlayer(InputStream stream, String type)</code>. The type parameter represents the input media content type. 2. Check for a <code>MediaException</code> if null is the content type. <pre>RecordStore recSt; int recId; try { InputStream inpStr = new ByteArrayInputStream((store.getRecord(recId))); Player p = Manager.createPlayer(inpStr, "audio/mpeg"); p.start(); } catch (IOException ioEx) { } catch (MediaException meEx) {}</pre>
Create a player for streaming media.	<p>For BlackBerry devices that operate on EDGE networks, Real Time Streaming Protocol (RTSP) functionality is available only over a Wi-Fi® connection.</p> <ul style="list-style-type: none"> > Invoke <code>Manager.createPlayer(String locator)</code>, passing an RTSP locator as a parameter. <pre>Manager.createPlayer("rtsp://streaming.rim.com/streaming_video.3gp");</pre>

Task	Steps
Create a player that displays a video in a field.	<ol style="list-style-type: none"> 1. Create <code>Player</code>, <code>VideoControl</code>, and <code>Field</code> variables. <pre>Player _videoPlayer; VideoControl _videoControl; Field videoField;</pre> 2. Start a try block. <pre>try {</pre> 3. Invoke <code>Manager.createPlayer(String locator)</code>, where <code>locator</code> is a string in URI syntax that describes the video content. Store a reference to the <code>Player</code> object that the call to <code>createPlayer(String locator)</code> returns. <pre>_videoPlayer = Manager.createPlayer("file:///SDCard/BlackBerry/videos/soccer1.avi");</pre> 4. To enable a <code>Player</code> to get the information it requires to acquire media resources, invoke <code>Player.realize()</code>. <pre>_videoPlayer.realize();</pre> 5. Invoke <code>Player.getControl()</code>, using as a parameter a string representation of the <code>VideoControl</code> class. Cast the returned object as a <code>VideoControl</code> object. <pre>_videoControl = (VideoControl)_videoPlayer.getControl("javax.microedition.media.control.VideoControl");</pre> 6. To initialize the mode that a <code>videoField</code> uses to display the video, invoke <code>VideoControl.initDisplayMode(int mode, Object arg)</code>. Use the <i>arg</i> parameter to specify the UI primitive that will display the video. For example, in a BlackBerry Application, use <code>"net.rim.device.api.ui.Field"</code> as the <i>arg</i> parameter, casting the object that this method returns as a <code>Field</code> object. See the API reference for the BlackBerry Java Development Environment for more information. <pre>videoField = (Field)_videoControl.initDisplayMode(VideoControl.USE_GUI_PRIMITIVE, "net.rim.device.api.ui.Field");</pre> 7. Check for any exceptions that may have occurred within the try block. <pre>} catch (Exception e) { System.out.println("Exception: " + e.toString()); }</pre>

Code fragment: Creating a player for a sequence of tones

Example: Sequence of tones

```
// "Mary Had A Little Lamb" has "ABAC" structure
// Use block to repeat "A" section

byte tempo = 30; // 30 x 4 = tempo of 120 bpm
byte duration = 8; // Note length 8 (quaver) = 1/8th of a note duration

byte C4 = ToneControl.C4; // C note value = 60 (middle C)
byte D4 = (byte)(C4 + 2); // D note value = 62 (a whole step)
byte E4 = (byte)(C4 + 4); // E note value = 64 (a major third)
byte G4 = (byte)(C4 + 7); // G note value = 67 (a fifth)
byte rest = ToneControl.SILENCE; // rest

byte[] mySequence = {
    ToneControl.VERSION, 1,      // version 1
    ToneControl.TEMPO, tempo,    // set tempo
```

```

//
// Start define "A" section
ToneControl.BLOCK_START, 0,
//
// Content of "A" section
E4, duration, D4, duration, C4, duration, E4, duration,
E4, duration, E4, duration, E4, duration, rest, duration,
//
// End define "A" section
ToneControl.BLOCK_END, 0,
//
// Play "A" section
ToneControl.PLAY_BLOCK, 0,
//
// Play "B" section
D4, duration, D4, duration, D4, duration, rest, duration,
E4, duration, G4, duration, G4, duration, rest, duration,
//
// Repeat "A" section
ToneControl.PLAY_BLOCK, 0,
//
// Play "C" section
D4, duration, D4, duration, E4, duration, D4, duration, C4, duration
};

try{
    Player p = Manager.createPlayer(Manager.TONE_DEVICE_LOCATOR);
    p.realize();
    ToneControl c = (ToneControl)p.getControl("ToneControl");
    c.setSequence(mySequence);
    p.start();
} catch (IOException ioe) {
} catch (MediaException me) { }

```

Code fragment: Creating a player for media from an input stream

Example: Play an MP3 audio file

```

//First we determine the supported content types
String types[] = Manager.getSupportedContentTypes(null);
for (int cnt = types.length - 1; cnt >= 0; --cnt) {

    if (types[cnt].equals("audio/mpeg")) {

        try {
            //Retrieve the MP3 file
            Class clazz = Class.forName("com.rim.samples.AudioDemo");
            InputStream is = clazz.getResourceAsStream("/ TarzanYell.mp3");

            //Create an instance of the player from the InputStream
            Player player = javax.microedition.media.Manager.createPlayer
            (is, "audio/mpeg");

            player.realize();
            player.prefetch();
        }
    }
}

```

```

        //start the player
        player.start();
    } catch (Exception ex) { }
}

else if (types[cnt].equals("audio/x-wav ")) {
    //this is where you would play wav files
}

else if (types[cnt].equals("audio/midi ")) {
    //this is where you would play midi files
}
}
}

```

Play media

You can use the API items in the `javax.microedition.media` package (JSR 135) to create a BlackBerry® Java® Application that can play media.

Task	Steps
Prepare the media player.	<ol style="list-style-type: none"> 1. Invoke <code>Player.realize()</code>. 2. Invoke <code>Player.prefetch()</code>.
Start the media player.	<p>> Invoke <code>Player.start()</code>. The <code>Player</code> returns to the <code>Prefetched</code> state when you invoke <code>Player.stop()</code> or when it reaches the end of the media file.</p> <pre> try { Player p = Manager.createPlayer("http://www.test.rim.net/abc.wav"); p.start(); } catch (MediaException pe) { } catch (IOException ioe) { } </pre>
Determine the controls that a media player supports.	<ol style="list-style-type: none"> 1. Invoke <code>Player.getControls()</code>. 2. To provide additional functionality for a media player, use one or more of the controls that the media player supports. You can use the same object to access multiple controls: for example, one object can be both a <code>VolumeControl</code> and a <code>ToneControl</code>. The <code>javax.microedition.media</code> package contains a number of Control interfaces. See the <i>API Reference</i> in the BlackBerry® Java® Development Environment for more information about the <code>javax.microedition.media</code> package.

Task	Steps
Enable video playback support	<ol style="list-style-type: none"> 1. Invoke <code>Player.getControls()</code> to retrieve a <code>VideoControl</code> object. 2. Implement the methods of the <code>VideoControl</code> interface to give a BlackBerry® Java® Application a variety of video support features, including the following: <ul style="list-style-type: none"> • control over the mode of video display (one of <code>USE_GUI_PRIMITIVE</code> or <code>USE_DIRECT_VIDEO</code>) • control over the location of the video with respect to the canvas that displays the video • access to the X-coordinate and the y-coordinate of the video with respect to the GUI object that displays the video • displaying or hiding video • resizing the video image
Adjust the volume of the media player.	<ol style="list-style-type: none"> 1. Invoke <code>VolumeControl()</code>. 2. Define a volume value in the following range: <ul style="list-style-type: none"> • 0: no volume • 100: maximum volume level <p>The <code>PlayerListener</code> sends a <code>VOLUME_CHANGED</code> event when its state changes.</p>
Close the media player.	> Invoke <code>Player.stop()</code> .

Listen for media player events

You can use the API items in the `javax.microedition.media` package (JSR 135) to create a BlackBerry® Java® Application that can listen for and send media player events.

Task	Steps
Listen for changes to the media player state.	<ol style="list-style-type: none"> 1. Implement <code>PlayerListener</code>. 2. To register the player listener, invoke <code>addPlayerListener</code>. <pre>private void doPlay() throws IOException, MediaException {Player p = Manager.createPlayer("http://www.rim.com/rim.mp3"); p.addPlayerListener(this); p.realize(); p.prefetch(); p.start(); }</pre>
Send a media player event to a registered player listener.	<pre>> Invoke playerUpdate(Player player, String event, Object eventData). public void playerUpdate(Player player, String event, Object eventData) { // Release resources player.close(); if (event == PlayerListener.END_OF_MEDIA) // Add code for actions if the end of media is reached. }</pre>

Using rich media

Playing rich media content

To play rich media content, use the following classes:

- To retrieve PME content on BlackBerry® devices or networks, use methods from the `MediaManager` class.
- To play PME content that exists on BlackBerry devices, use methods from the `MediaPlayer` class.



Note: To display content created in Plazmic® Media Engine Version 4.2, the screen must not support scrolling. See "Playing rich media content" on page 55 for more information on how to create a screen that does not support scrolling.

Download rich media content

1. Create a `MediaManager` object.
2. Invoke `MediaManager.createMedia()`.

The first time that you invoke `MediaManager.createMedia()`, the URI must be absolute, unless you first invoke `MediaManager.setProperty("URI_BASE", base_url)` to set a base URL. When you invoke `createMedia()` subsequently, the URL that the method used previously is the base.

```
MediaManager manager = new MediaManager();
try {
Object media = manager.createMedia("http://webserver/sample.pme");
} catch (IOException ioe) {
System.out.println("Error: requested content was not downloaded.");
} catch (MediaException me) {
System.out.println("Error: " + me.getCode()); }
```

Play rich media content

Task	Steps
Set the PME object for playback.	<ul style="list-style-type: none"> > Invoke <code>MediaPlayer.setMedia()</code>. <pre>MediaPlayer player = new MediaPlayer(); try { player.setMedia(media); } catch (MediaException me) { System.out.println("Error: requested content type is not supported."); }</pre>
Allow an application's screen to display content created in Plazmic Media Engine Version 4.2.	<p>To display content created in Plazmic Media Engine Version 4.2, the screen must not support scrolling.</p> <ul style="list-style-type: none"> > Create an instance of a screen object using the <code>NO_VERTICAL_SCROLL</code> and <code>NO_HORIZONTAL_SCROLL</code> fields (inherited from the <code>Manager</code> class). <pre>Screen screen = new Screen(Screen.NO_VERTICAL_SCROLL Screen.NO_HORIZONTAL_SCROLL);</pre>
Retrieve a UI object that displays rich media content.	<ol style="list-style-type: none"> 1. Invoke <code>MediaPlayer.getUI()</code>. 2. Cast the object that <code>getUI()</code> returns as a <code>Field</code>, and add it to a <code>Screen</code> for display. <pre>screen.add((Field)player.getUI());</pre>

Task	Steps
Play rich media content.	<ol style="list-style-type: none">1. Check the media player state.2. Invoke <code>MediaPlayer.start()</code>. <pre>if(player.getState() == MediaPlayer.REALIZED) { try { player.start(); } catch(MediaException me) { System.out.println("Error occurred during media playback: " + me.getCode() + me.getMessage()); } }</pre>

See “Code sample: Retrieving and displaying a rich media file” on page 63 for more information.

Listen for rich media events

Task	Steps
Listen for media engine events.	<ol style="list-style-type: none"> 1. Implement the <code>MediaListener</code> interface to let your BlackBerry® Java® Application listen for media engine events. 2. Implement <code>mediaEvent()</code> to handle all possible media events. <pre> public final class MediaListenerImpl implements MediaListener { public void mediaEvent(Object sender, int event, int eventParam, Object data) { switch(event) { case MEDIA_REQUESTED: // Perform action. break; case MEDIA_COMPLETE: // Perform action. break; case MEDIA_REALIZED: // Perform action. break; case MEDIA_IO: // Perform action. break; } } } </pre>
Register the listener.	<pre> > Invoke addMediaListener() on the MediaPlayer and MediaManager objects. private MediaListenerImpl _listener = new MediaListenerImpl(); private MediaPlayer player = new MediaPlayer(); private MediaManager manager = new MediaManager(); player.addMediaListener(_listener); manager.addMediaListener(_listener); </pre>

Task	Steps
Load content in the background, and play it when the download is complete.	<ol style="list-style-type: none"> 1. To download content for future playback, invoke <code>MediaManager.createMediaLater()</code>. 2. In <code>MediaListener.mediaEvent()</code>, add code to manage the <code>MEDIA_REALIZED</code> event that occurs when the content the application downloads finishes loading on the BlackBerry® device. 3. To register the content that the data parameter specifies, invoke <code>MediaPlayer.setMedia(data)</code>. 4. To start playback, invoke <code>MediaPlayer.start()</code>. <pre> manager.createMediaLater("http://webserver/sample.pme"); public void mediaEvent(Object sender, int event, int eventParam, Object data) { switch(event) { ... case MEDIA_REALIZED: try { player.setMedia(data); player.start(); } catch(MediaException me) { System.out.println("Error playing media" + me.getCode() + me.getMessage()); } break; } } } </pre>
Track the progress of a download.	<ol style="list-style-type: none"> 1. Extend the <code>net.rim.plazmic.mediaengine.io.LoadingStatus</code> class. 2. In your implementation of <code>mediaEvent()</code>, when the <code>MEDIA_IO</code> event occurs, cast the <code>Object</code> in the data parameter to a <code>LoadingStatus</code> object. 3. To retrieve the download status, and manage each status, invoke <code>LoadingStatus.getStatus()</code>. 4. For each normal status, print a message to the console.
Manage a failed download.	<p>For the <code>LOADING_FAILED</code> status, perform the following actions:</p> <ol style="list-style-type: none"> 1. To retrieve the error code, invoke <code>LoadingStatus.getCode()</code>. 2. To retrieve the detailed message, invoke <code>LoadingStatus.getMessage()</code>. 3. To retrieve the URL string of the content, invoke <code>LoadingStatus.getSource()</code>.

Code fragment: Managing rich media content download events

Example: Code fragment: Managing rich media content download events

```

public void mediaEvent(Object sender, int event, int eventParam, Object data) {
    switch(event) {
        ...
        case MEDIA_IO: {
            LoadingStatus s = (LoadingStatus)data;
        }
        ...
        break;
    }
    break;
    ...
    switch(s.getStatus()) {
        case LoadingStatus.LOADING_STARTED:

```

```

        System.out.println("Loading in progress");
        break;
        case LoadingStatus.LOADING_READING:
            System.out.println("Parsing in progress");
            break;
        case LoadingStatus.LOADING_FINISHED:
            System.out.println("Loading completed");
            break;
        case LoadingStatus.LOADING_FAILED:
            String errorName = null;
            int code = s.getCode();
            switch (code) {
                case MediaException.INVALID_HEADER:
                    errorName = "Invalid header" + "\n" + s.getSource();
                    break;
                case MediaException.REQUEST_TIMED_OUT:
                    errorName = "Request timed out" + "\n" + s.getSource();
                    break;
                case MediaException.INTERRUPTED_DOWNLOAD:
                    break;
                case MediaException.UNSUPPORTED_TYPE:
                    errorName = "Unsupported type" + s.getMessage() + "\n" + s.getSource();
                    break;
                default: {
                    if (code > 200) {
                        // A code > 200 indicates an HTTP error
                        errorName = "URL not found";
                    } else {
                        // default unidentified error
                        errorName = "Loading Failed";
                    }
                    errorName += "\n" + s.getSource() + "\n" + s.getCode()
                        + ": " + s.getMessage();
                    break;
                }
            }
        }
        System.out.println(errorName);
        break;
    } // End switch s.getStatus().
    break;
}

```

See "Code sample: Implementing a listener to download rich media content" on page 64 for more information.

Create a custom connector for rich media connections

To add support for a custom protocol or to override default behavior, create a custom `Connector`.

Task	Steps
Implement a custom connector.	> Implement the <code>net.rim.plazmic.mediaengine.io.Connector</code> interface.
Return an input stream to read content from a URL.	> Implement <code>InputStream getInputStream(String, ConnectionInfo)</code> .
Set custom connector properties.	> Implement <code>void setProperty(String, String)</code> .

Task	Steps
Release the custom connection.	> Implement <code>void releaseConnection(ConnectionInfo)</code> .
Register a custom connector.	> In your main method, invoke <code>MediaManager.setConnector()</code> . <pre>MediaManager manager = new MediaManager(); manager.setConnector(new CustomPMEConnector(manager.getDefaultConnector()));</pre>

See "Code sample: Implementing a custom connector" on page 66 for more information.

Code samples

Code sample: Using a raw image to recreate an encoded image

Example: ImageDemo.java

```
/**
 * ImageDemo.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */
package com.rim.samples.docs.imagedemo;

import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import java.io.*;

/* The ImageDemo.java sample retrieves raw data from an image that
   is included in its project, and then uses that raw data to
   recreate an EncodedImage. */
public class ImageDemo extends UiApplication {
    public static void main(String[] args) {
        ImageDemo app = new ImageDemo();
        app.enterEventDispatcher();
    }
    public ImageDemo() {
        pushScreen(new ImageDemoScreen());
    }
}

final class ImageDemoScreen extends MainScreen {
    private static final int IMAGE_SIZE = 2430;
    private InputStream input;
    private byte[] data = new byte[IMAGE_SIZE];

    public ImageDemoScreen() {
        super();
        setTitle(new LabelField("Image Demo Sample"));

        try {
```

```

        input =
Class.forName("com.rim.samples.docs.imagedemo.ImageDemo").getResourceAsStream("/images/
hellokitty.png");
    } catch (ClassNotFoundException e) {
        System.out.println("Class not found");
    }

    if(input == null) {
        System.out.println("Error: input stream is not initialized.");
    } else if (input != null) {
        System.out.println("OK: input stream is initialized.");
        try {
            int code = input.read(data);
            System.out.println("Total number of bytes read into buffer: " + code + "
.");
        } catch (IOException e) {
            // Handle exception.
        }

        try {
            EncodedImage image = EncodedImage.createEncodedImage(data, 0, data.length);
            add(new BitmapField(image.getBitmap()));
        } catch (IllegalArgumentException iae) {
            System.out.println("Image format not recognized.");
        }
    }
}
}
}

```

Code sample: Drawing a new bitmap using an existing bitmap

To draw a new bitmap image, the DrawDemo.java sample retrieves raw data from a predefined bitmap image. It then displays the original and restored images.

Example: DrawDemo.java

```

/*
 * DrawDemo.java
 * Copyright (C) 2002-2005 Research In Motion Limited.
 */

package com.rim.samples.docs.drawing;

import net.rim.device.api.system.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;

/* The DrawDemo.java sample retrieves raw data from a predefined bitmap
   image, and then draws a new bitmap using the data. It then displays
   the original and restored images. */
public class DrawDemo extends UiApplication {

```

```

    public static void main(String[] args) {
        DrawDemo app = new DrawDemo();
        app.enterEventDispatcher();
    }

    public DrawDemo() {
        pushScreen(new DrawDemoScreen());
    }
}

final class DrawDemoScreen extends MainScreen {
    public DrawDemoScreen() {
        super();

        LabelField title = new LabelField("UI Demo", LabelField.USE_ALL_WIDTH);
        setTitle(title);

        Bitmap original = Bitmap.getPredefinedBitmap(Bitmap.INFORMATION);
        Bitmap restored = new Bitmap(original.getType(), original.getWidth(),
            original.getHeight());

        Graphics graphics = new Graphics(restored);

        // Retrieve raw data from original image.
        int[] argb = new int[original.getWidth() * original.getHeight()];
        original.getARGB(argb, 0, original.getWidth(), 0, 0, original.getWidth(),
            original.getHeight());

        // Draw new image using raw data retrieved from original image.
        try {
            graphics.drawRGB(argb, 0, restored.getWidth(), 0, 0, restored.getWidth(),
                restored.getHeight());
        } catch (Exception e) {
            System.out.println("Error occurred during drawing: " + e);
        }

        if(restored.equals(original)) {
            System.out.println("Success! Bitmap renders correctly with RGB data.");
        } else if(!restored.equals(original)) {
            System.out.println("Bitmap rendered incorrectly with RGB data.");
        }

        BitmapField field1 = new BitmapField(original, BitmapField.STAMP_MONOCHROME);
        BitmapField field2 = new BitmapField(restored);
        add(new LabelField("Original bitmap: "));
        add(field1);
        add(new LabelField("Restored bitmap: "));
        add(field2);
    }
}

```

Code sample: Retrieving and displaying a rich media file

The MediaSample.java sample retrieves a .pme file from a web server and displays it on the BlackBerry® device.

Example: MediaSample.java

```
/**
 * MediaSample.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.mediasample;

import java.io.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;

import net.rim.plazmic.mediaengine.*;

public class MediaSample extends UiApplication {

    public static void main(String[] args) {
        MediaSample app = new MediaSample();
        app.enterEventDispatcher();
    }

    public MediaSample() {
        pushScreen(new MediaSampleScreen());
    }

    final static class MediaSampleScreen extends MainScreen {
        public MediaSampleScreen() {
            super();
            LabelField title = new LabelField("Media Sample", LabelField.ELLIPSIS
            | LabelField.USE_ALL_WIDTH);
            setTitle(title);

            MediaPlayer player = new MediaPlayer();
            MediaManager manager = new MediaManager();

            try {
                Object media = manager.createMedia("http://webserver/SVGFILE.pme");
                player.setMedia(media);
            } catch (IOException ioe) {
            } catch (MediaException me) {
                System.out.println("Error during media loading: ");
                System.out.println(me.getCode());
                System.out.println(me.getMessage());
            }
            add((Field)player.getUI());
            try {
                player.start();
            } catch (MediaException me) {
                System.out.println("Error occurred during media playback: ");
                System.out.println(me.getCode());
                System.out.println(me.getMessage());
            }
        }
    }
}
```

```

    }
}

```

Code sample: Implementing a listener to download rich media content

The `MediaSample2.java` sample implements a listener to download media content in the background and display the download status to the console.

Example: `MediaSample2.java`

```

/**
 * MediaSample2.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.mediasample;

import java.io.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;

import net.rim.plazmic.mediaengine.*;
import net.rim.plazmic.mediaengine.io.*;

public class MediaSample2 extends UiApplication {
    private MediaPlayer player = new MediaPlayer();
    private MediaManager manager = new MediaManager();
    private MediaListenerImpl _listener = new MediaListenerImpl();

    private MediaSample2Screen _screen;

    public static void main(String[] args) {
        MediaSample2 app = new MediaSample2();
        app.enterEventDispatcher();
    }

    public MediaSample2() {
        _screen = new MediaSample2Screen();
        pushScreen(_screen);
    }

    public final class MediaListenerImpl implements MediaListener {
        public void mediaEvent(Object sender, int event, int eventParam, Object data) {
            switch(event) {
                case MEDIA_REQUESTED:
                    System.out.println("Media requested");
                    break;
                case MEDIA_COMPLETE:
                    System.out.println("Media completed");
                    break;
                case MEDIA_REALIZED:
                    try {

```

```

        player.setMedia(data);
        player.start();
    }
    catch(MediaException me) {
        System.out.println("Error during media loading: " + me.getCode() +
me.getMessage());
    }
    break;
case MEDIA_IO: {
    LoadingStatus s = (LoadingStatus)data;
    switch(s.getStatus()) {
        case LoadingStatus.LOADING_STARTED:
            System.out.println("Loading in progress");
            break;
        case LoadingStatus.LOADING_READING:
            System.out.println("Parsing in progress");
            break;
        case LoadingStatus.LOADING_FINISHED:
            System.out.println("Loading completed");
            break;
        case LoadingStatus.LOADING_FAILED:
            String errorName = null;
            int code = s.getCode();
            switch (code) {
                case MediaException.INVALID_HEADER:
                    errorName = "Invalid header" + "\n" + s.getSource();
                    break;

                case MediaException.REQUEST_TIMED_OUT:
                    errorName = "Request timed out" + "\n" + s.getSource();
                    break;

                case MediaException.INTERRUPTED_DOWNLOAD:
                    break;

                case MediaException.UNSUPPORTED_TYPE:
                    errorName = "Unsupported type" + s.getMessage() + "\n"
+ s.getSource();
                    break;

                default: {
                    if (code > 200) {
                        // A code > 200 indicates an HTTP error.
                        errorName = "URL not found";
                    }
                    else {
                        // Default unidentified error.
                        errorName = "Loading Failed";
                    }
                    errorName += "\n" + s.getSource() + "\n"
+ s.getCode() + ": " + s.getMessage();
                    break;
                }
            }
            System.out.println(errorName);
            break;
    } // End switch s.getStatus().

```

```

        break;
    }
}
}
}

final class MediaSample2Screen extends MainScreen {
    public MediaSample2Screen() {
        super();
        LabelField title = new LabelField("Media Sample", LabelField.ELLIPSIS
            | LabelField.USE_ALL_WIDTH);
        setTitle(title);

        player.addMediaListener(_listener);
        manager.addMediaListener(_listener);

        // Change this to the location of a test .pme file.
        manager.createMediaLater("http://test.rim.com/SVGBS0001.pme");
        add((Field)player.getUI());
    }
}
}

```

Code sample: Implementing a custom connector

The CustomPMEConnector.java sample provides a framework for implementing a custom connector.

Example: CustomPMEConnector.java

```

/*
 * CustomPMEConnector.java
 * Copyright (C) 2003-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.mediasample;

import java.io.*;
import net.rim.plazmic.mediaengine.*;
import net.rim.plazmic.mediaengine.io.*;

public class CustomPMEConnector implements Connector {

    private Connector delegate;
    private InputStream input;

    CustomPMEConnector(Connector delegate) {
        this.delegate = delegate;
    }

    public InputStream getInputStream(String uri, ConnectionInfo info)
        throws IOException, MediaException {
        if (uri.startsWith("myprotocol://")) {
            // Perform special tasks.
            info.setConnection(new MyProtocolConnection());
            info.setContentType("application/x-vnd.rim.pme");
            // OpenMyInputStream() is a custom method that opens

```

```

        //stream for "myprotocol://"
        input = openMyInputStream(uri);
    } else {
        input = delegate.getInputStream(uri, info);
    }
    return input;
}

private InputStream openMyInputStream(String uri) {
    InputStream input = null;
    // @todo: open stream here
    return input;
}

public void releaseConnection(ConnectionInfo info)
    throws IOException, MediaException {
    Object o = info.getConnection();
    if (o instanceof MyProtocolConnection) {
        ((MyProtocolConnection)o).close(); // Perform cleanup.
    } else {
        delegate.releaseConnection(info);
    }
}

public void setProperty(String property, String value) {
    delegate.setProperty(property, value);
}

// Inner class that defines the connection class.
public static class MyProtocolConnection {
    public MyProtocolConnection() {
        // ...
    }
    public void close() {
        // ...
    }
}
}

```

Storing data

[Use BlackBerry persistent storage](#)
[Manage persistent data](#)
[Manage custom objects](#)
[Use the MIDP record store](#)
[Code samples](#)

Use BlackBerry persistent storage

Storage method	Description
BlackBerry® persistence model	<p>The BlackBerry persistence model provides a flexible and efficient way to store data. When writing a BlackBerry Java Application specifically for BlackBerry devices, use the BlackBerry persistence model.</p> <ul style="list-style-type: none"> The BlackBerry persistence model lets you save any <code>Object</code> in the persistent store. As a result, searching for data in the persistent store is faster than searching in the record store model. To store custom object types, the class of the custom type must use the <code>Persistable</code> interface. In the BlackBerry persistence model, BlackBerry Java Applications can share data at the discretion of the BlackBerry Java Application that creates the data. Code signing specifies that only authorized BlackBerry Java Applications can access the data.
MIDP record stores	<p>The MIDP record store allows a BlackBerry Java Application to be portable across multiple devices that are compatible with the Java Platform, Micro Edition.</p> <ul style="list-style-type: none"> In MIDP, store persistent data as records in <code>RecordStore</code> objects. MIDP records store data only as byte arrays. In MIDP 2.0 and later, if an application creates a record store using the <code>RecordStore.AUTHMODE_ANY</code> field, a MIDlet suite can share the record store with other MIDlet suites. See the API reference for the BlackBerry Java Development Environment for more information about the <code>RecordStore</code> class.

BlackBerry persistent storage

Feature	Description
Security	By default, BlackBerry® Java® Applications on the BlackBerry device that are digitally signed by Research In Motion can access the data in the persistent store. Contact RIM for information on controlling access to the data.
Administrative control	<p>With the BlackBerry® Enterprise Server, system administrators can use IT policies to control the use of persistent storage by third-party BlackBerry Java® Applications.</p> <p>Administrators can set <code>ALLOW_USE_PERSISTENT_STORE</code> to <code>TRUE</code> or <code>FALSE</code>. By default, third-party BlackBerry Java Applications are enabled to use persistent storage (<code>ALLOW_USE_PERSISTENT_STORE</code> is <code>TRUE</code>). This policy does not affect the MIDP record store.</p>

Feature	Description
Data integrity	<p>To maintain the integrity of data in persistent storage, partial updates are not made if an error occurs during a commit. Data in the <code>PersistentObject</code> retains the values from the last commit in order to preserve data integrity.</p> <p>If the BlackBerry JVM performs an emergency garbage collection operation due to low memory, outstanding transactions are committed immediately to avoid compromising data integrity. If the device fails during this operation, partially completed transactions are committed when the BlackBerry device starts. Outstanding transactions are not committed during normal garbage collection operation.</p>

Manage persistent data

Task	Steps
Create a unique long key.	<p>Each <code>PersistentObject</code> has a unique long key.</p> <ol style="list-style-type: none"> 1. In the BlackBerry® Integrated Development Environment, type a string value, such as <code>com.rim.samples.docs.userinfo</code>. 2. Select this string. 3. Right-click this string and click Convert 'com.rim.samples.docs.userinfo' to long. 4. Include a comment in your code to indicate the string that you used to generate the unique long key.
Create a persistent data store.	<ol style="list-style-type: none"> 1. Create a single static <code>PersistentObject</code>. 2. Invoke <code>PersistentStore.getPersistentObject</code>, using the unique long key as a parameter. <pre>static PersistentObject store; static { store = PersistentStore.getPersistentObject(0xa1a569278238dad2L); }</pre>
Store an object persistently.	<ol style="list-style-type: none"> 1. Invoke <code>setContents()</code> on a <code>PersistentObject</code>. This method replaces existing content with the new content. 2. To save the new content to the persistent store, invoke <code>commit()</code>. <pre>String[] userinfo = {username, password}; synchronized(store) { store.setContents(userinfo); store.commit(); }</pre>
Store objects in a batch transaction.	<ol style="list-style-type: none"> 1. To use a batch transaction to commit objects to the persistent store, invoke <code>PersistentStore.getSyncObject()</code>. This method retrieves the persistent store monitor that locks the object. 2. Synchronize on the object. 3. Invoke <code>commit()</code> as necessary. If any commit in the batch fails, the entire batch transaction fails.
Commit a monitor object separately from a batch transaction.	<p>> Invoke <code>forceCommit()</code> while synchronizing the monitor object.</p>

Task	Steps
Retrieve persistent data.	<ol style="list-style-type: none"> 1. Invoke <code>getContents()</code> on a <code>PersistentObject</code>. 2. To convert to your desired format, perform an explicit cast on the object that <code>PersistentObject.getContents()</code> returns. <pre>synchronized(store) { String[] currentinfo = (String[])store.getContents(); if(currentinfo == null) { Dialog.alert(_resources.getString(APP_ERROR)); } else { currentusernamefield.setText(currentinfo[0]); currentpasswordfield.setText(currentinfo[1]); } }</pre>
Remove all persistent data from an BlackBerry® Java® Application.	<p>If you delete the .cod file that defines a <code>PersistentStore</code>, then all persistent objects that the .cod file created are deleted.</p> <ul style="list-style-type: none"> > Invoke <code>PersistentStore.destroyPersistentObject()</code>, providing as a parameter a unique key for the <code>PersistentObject</code>.
Remove specific persistent data from a BlackBerry® Java® Application.	<ul style="list-style-type: none"> > To delete individual data, treat the data as normal objects, and remove references to it. A garbage collected operation removes the data.

See "Code sample: Saving user name and password information" on page 75 for more information.

Manage custom objects

Task	Steps
Create an object to store data.	<ol style="list-style-type: none"> 1. Create a <code>Vector</code> object in which to store multiple objects. 2. Create a single static <code>PersistentObject</code>. <pre>private static Vector _data; PersistentObject store; static { store = PersistentStore.getPersistentObject(0xdec6a67096f833cL); //key is hash of test.samples.restaurants _data = (Vector)store.getContents(); synchronized (store) { if (_data == null) { _data = new Vector(); store.setContents(_data); store.commit(); } } }</pre>
Store data persistently.	<p>> In the class for the objects that you want to store, implement the <code>Persistable</code> interface.</p> <pre>private static final class RestaurantInfo implements Persistable { private String[] _elements; public static final int NAME = 0; public static final int ADDRESS = 1; public static final int PHONE = 2; public static final int SPECIALTY = 3; public RestaurantInfo() { _elements = new String[4]; for (int i = 0; i < _elements.length(); ++i) { _elements[i] = new String(""); } } public String getElement(int id) { return _elements[id]; } public void setElement(int id, String value) { _elements[id] = value; } }</pre>

Task	Steps
Save an object.	<ol style="list-style-type: none"> 1. Define an object. The following code fragment creates a <code>RestaurantInfo</code> object and uses its set methods to define its components. <pre> RestaurantInfo info = new RestaurantInfo(); info.setElement(RestaurantInfo.NAME, namefield.getText()); info.setElement(RestaurantInfo.ADDRESS, addressfield.getText()); info.setElement(RestaurantInfo.PHONE, phonefield.getText()); info.setElement(RestaurantInfo.SPECIALTY, specialtyfield.getText()); </pre> 2. Add the object to a vector by invoking <code>addElement()</code>. <pre> _data.addElement(info); </pre> 3. Synchronize with the persistent object so that other threads cannot make changes to the object at the same time. <pre> synchronized(store) { </pre> 4. Invoke <code>setContents()</code>. <pre> store.setContents(_data); </pre> 5. To save the updated data, invoke <code>commit()</code> on the <code>PersistentObject</code>. <pre> store.commit(); } </pre>
Retrieve the most recently saved object.	<pre> > Invoke _data.lastElement(). public void run() { synchronized(store) { _data = (Vector)store.getContents(); if (!_data.isEmpty()) { RestaurantInfo info = (RestaurantInfo)_data.lastElement(); namefield.setText(info.getElement(RestaurantInfo.NAME)); addressfield.setText(info.getElement(RestaurantInfo.ADDRESS)); phonefield.setText(info.getElement(RestaurantInfo.PHONE)); specialtyfield.setText(info.getElement(RestaurantInfo.SPECIALTY));} } } </pre>

See "Code sample: Storing and viewing restaurant information" on page 77 for more information.

Use the MIDP record store

Task	Steps
Create a record store.	<p>> Invoke <code>openRecordStore()</code>, and specify <code>true</code> to indicate that the method should create the record store if the record store does not exist.</p> <pre>RecordStore store = RecordStore.openRecordStore("Contacts", true);</pre>
Add a record.	<p>> Invoke <code>addRecord()</code>.</p> <pre>int id = store.addRecord(_data.getBytes(), 0, _data.length());</pre>
Retrieve a record.	<p>> Invoke <code>getRecord(int, byte[], int)</code>, and provide the following parameters:</p> <ul style="list-style-type: none"> • a record ID • a byte array • an offset <pre>byte[] data = new byte[store.getRecordSize(id)]; store.getRecord(id, data, 0); String dataString = new String(data);</pre>
Retrieve all records.	<ol style="list-style-type: none"> 1. Invoke <code>openRecordStore()</code>. 2. Invoke <code>enumerateRecords()</code> with the following parameters: <ul style="list-style-type: none"> • <code>filter</code>: specifies a <code>RecordFilter</code> object to retrieve a subset of record store records (if null, the method returns all records) • <code>comparator</code>: specifies a <code>RecordComparator</code> object to determine the order in which the method returns the records (if null, the method returns the records in any order) • <code>keepUpdated</code>: determines if the method keeps the enumeration current with the changes to the record store <pre>RecordStore store = RecordStore.openRecordStore("Contacts", false); RecordEnumeration e = store.enumerateRecords(null, null, false);</pre>

Code samples

Code sample: Saving user name and password information

This code sample demonstrates how to create a BlackBerry® Java® Application for BlackBerry device users to view their current user names and passwords, type new user names and passwords and save changes.

Example: UserInfo.java

```
/**
 * UserInfo.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.userinfo;

import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;
```

```

import java.util.*;
import net.rim.device.api.i18n.*;
import com.rim.samples.docs.resource.*;

public class UserInfo extends UiApplication implements UserInfoResource
{
    private static PersistentObject store;
    private static ResourceBundle _resources;
    private AutoTextEditField usernamefield;
    private PasswordEditField passwordfield;
    private AutoTextEditField currentusernamefield;
    private AutoTextEditField currentpasswordfield;

    static {
        _resources = ResourceBundle.getBundle(
            UserInfoResource.BUNDLE_ID, UserInfoResource.BUNDLE_NAME);
        store = PersistentStore.getPersistentObject(0xa1a569278238dad2L);
    }

    private MenuItem saveItem = new MenuItem( _resources.getString(MENUITEM_SAVE), 110, 10)
    {
        public void run() {
            String username = usernamefield.getText();
            String password = passwordfield.getText();
            String[] userinfo = {username, password};
            synchronized(store) {
                store.setContents(userinfo);
                store.commit();
            }

            Dialog.inform(_resources.getString(APP_SUCCESS));

            usernamefield.setText(null);
            passwordfield.setText(null);
        }
    };

    private MenuItem getItem = new MenuItem( _resources.getString(MENUITEM_GET), 110, 11 )
    {
        public void run() {
            synchronized(store) {
                String[] currentinfo = (String[])store.getContents();
                if(currentinfo == null) {
                    Dialog.alert(_resources.getString(APP_ERROR));
                } else {
                    currentusernamefield.setText(currentinfo[0]);
                    currentpasswordfield.setText(currentinfo[1]);
                }
            }
        }
    };

    public static void main(String[] args) {
        UserInfo app = new UserInfo();
        app.enterEventDispatcher();
    }

    public UserInfo() {

```

```

MainScreen mainScreen = new UserMainScreen();
mainScreen.setTitle(new LabelField(
    _resources.getString(APPLICATION_TITLE)));

usernamefield = new AutoTextEditField(
    _resources.getString(FIELD_NAME), "");
passwordfield = new PasswordEditField(
    _resources.getString(FIELD_PASSWORD), "");
currentusernamefield = new AutoTextEditField(
    _resources.getString(FIELD_CURRENTNAME), "");
currentpasswordfield = new AutoTextEditField(
    _resources.getString(FIELD_CURRENTPASSWORD), "");

SeparatorField separator = new SeparatorField();

mainScreen.add(usernamefield);
mainScreen.add(passwordfield);
mainScreen.add(separator);
mainScreen.add(currentusernamefield);
mainScreen.add(currentpasswordfield);
pushScreen(mainScreen);
}

private final class UserMainScreen extends MainScreen
{
    protected void makeMenu( Menu menu, int instance ) {
        menu.add(saveItem);
        menu.add(getItem);
        super.makeMenu(menu, 0);
    }

    public void close() {
        Dialog.alert(_resources.getString(APP_EXIT));
        super.close();
    }
}
}

```

Code sample: Storing and viewing restaurant information

This code sample lets BlackBerry® device users save information about multiple restaurants and view information about the most recently saved restaurant.

Example: Restaurants.java

```

/**
 * Restaurants.java
 * Copyright (C) 2004-2005 Research In Motion Limited.
 */

package com.rim.samples.docs.restaurants;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;

```

```

import net.rim.device.api.util.*;
import java.util.*;
import net.rim.device.api.i18n.*;
import net.rim.blackberry.api.invoke.*;
import net.rim.blackberry.api.browser.*;
import com.rim.samples.docs.resource.*;

public class Restaurants extends UiApplication implements RestaurantResource {

    private AutoTextEditField namefield;
    private AutoTextEditField addressfield;
    private EditField phonefield;
    private EditField websitefield;
    private EditField specialtyfield;

    private static Vector _data;
    private static PersistentObject store;
    private static ResourceBundle _resources;

    private MenuItem saveItem = new MenuItem(_resources.getString(MENUITEM_SAVE), 110, 10)
{
    public void run() {
        RestaurantInfo info = new RestaurantInfo();

        info.setElement(RestaurantInfo.NAME, namefield.getText());
        info.setElement(RestaurantInfo.ADDRESS, addressfield.getText());
        info.setElement(RestaurantInfo.PHONE, phonefield.getText());
        info.setElement(RestaurantInfo.WEBSITE, phonefield.getText());
        info.setElement(RestaurantInfo.SPECIALTY,
specialtyfield.getText());

        _data.addElement(info);

        synchronized(store) {
            store.setContents(_data);
            store.commit();
        }
        Dialog.inform(_resources.getString(APP_SUCCESS));
        namefield.setText(null);
        addressfield.setText(null);
        phonefield.setText("");
        websitefield.setText("");
        specialtyfield.setText("");
    }
};

    private MenuItem getItem = new MenuItem(_resources.getString(MENUITEM_GET), 110, 11) {
        public void run() {
            synchronized(store) {
                _data = (Vector)store.getContents();
                if (!_data.isEmpty()) {
                    RestaurantInfo info = (RestaurantInfo)_data.lastElement();
                    namefield.setText(info.getElement(RestaurantInfo.NAME));
                    addressfield.setText(info.getElement(RestaurantInfo.ADDRESS));
                    phonefield.setText(info.getElement(RestaurantInfo.PHONE));
                    websitefield.setText(info.getElement(RestaurantInfo.WEBSITE));
                    specialtyfield.setText(info.getElement(RestaurantInfo.SPECIALTY));
                }
            }
        }
    }
};

```

```

    }
};

private MenuItem phoneItem = new MenuItem(_resources.getString(MENUITEM_PHONE), 110,
112) {
    public void run() {
        synchronized(store) {
            String phoneNumber = phonefield.getText();
            if ( phoneNumber.length() == 0) {
                Dialog.alert(_resources.getString(ALERT_NO_PHONENUMBER));
            } else {
                PhoneArguments call = new PhoneArguments(PhoneArguments.ARG_CALL,
phoneNumber);
                Invoke.invokeApplication(Invoke.APP_TYPE_PHONE, call);
            }
        }
    }
};

private MenuItem browserItem = new MenuItem(_resources.getString(MENUITEM_BROWSER),
110, 112) {
    public void run() {
        synchronized(store) {
            String websiteUrl = websitefield.getText();
            if (websiteUrl.length() == 0) {
                Dialog.alert(_resources.getString(ALERT_NO_WEBSITE));
            } else {
                BrowserSession visit = Browser.getDefaultSession();
                visit.displayPage(websiteUrl);
            }
        }
    }
};

static {
    _resources = ResourceBundle.getBundle(
        RestaurantResource.BUNDLE_ID,
        RestaurantResource.BUNDLE_NAME);
    store = PersistentStore.getPersistentObject(0xdec6a67096f833cL);
    // Key is hash of test.samples.restaurants.
    synchronized (store) {
        _data = (Vector)store.getContents();
        if (_data == null) {
            _data = new Vector();
            store.setContents( _data );
            store.commit();
        }
    }
}

public static void main(String[] args) {
    Restaurants app = new Restaurants();
    app.enterEventDispatcher();
}

private static final class RestaurantInfo implements Persistable {
    // Data.

```

```

private String[] _elements;

// Fields.
public static final int NAME = 0;
public static final int ADDRESS = 1;
public static final int PHONE = 2;
public static final int WEBSITE = 3;
public static final int SPECIALTY = 4;

public RestaurantInfo() {
    _elements = new String[4];
    for ( int i = 0; i < _elements.length; ++i) {
        _elements[i] = "";
    }
}

public String getElement(int id) {
    return _elements[id];
}

public void setElement(int id, String value) {
    _elements[id] = value;
}
}

private final class RestaurantsMainScreen extends MainScreen
{
    protected void makeMenu( Menu menu, int instance ) {
        menu.add(saveItem);
        menu.add(getItem);
        menu.add(phoneItem);
        menu.add(browserItem);
        super.makeMenu(menu, instance);
    }

    public void close() {
        Dialog.alert(_resources.getString(APP_EXIT));
        super.close();
    }
}

public Restaurants() {
    MainScreen mainScreen = new RestaurantsMainScreen();
    mainScreen.setTitle(new LabelField(
        _resources.getString(APPLICATION_TITLE)));
    namefield = new AutoTextEditField(
        _resources.getString(FIELD_NAME), "");
    addressfield = new AutoTextEditField(
        _resources.getString(FIELD_ADDRESS), "");
    phonefield = new EditField(
        _resources.getString(FIELD_PHONE), "", Integer.MAX_VALUE,
        BasicEditField.FILTER_PHONE);
    websitefield = new EditField(
        _resources.getString(FIELD_WEBSITE), "", Integer.MAX_VALUE,
        BasicEditField.FILTER_URL);
    specialtyfield = new EditField(
        _resources.getString(FIELD_SPECIALTY), "",
        Integer.MAX_VALUE, BasicEditField.FILTER_DEFAULT);
}

```

```
mainScreen.add(namefield);  
mainScreen.add(addressfield);  
mainScreen.add(phonefield);  
mainScreen.add(websitefield);  
mainScreen.add(specialtyfield);  
pushScreen(mainScreen);  
    }  
}
```

Managing data

Data synchronization
Backing up and restoring data
Code samples

Data synchronization

Research In Motion (RIM) does not provide tools or BlackBerry® Java® Applications for synchronizing data to remote data sources, so you must build the synchronization logic into your BlackBerry Java Application. See the *BlackBerry Java Development Environment Fundamentals Guide* for more information creating BlackBerry Java Applications for synchronizing data on a BlackBerry® device.

Types of data synchronization

Synchronization type	Description
Wireless (BlackBerry® Enterprise Server)	<p>The automatic wireless backup process on a BlackBerry Enterprise Server is designed to back up data from the BlackBerry device to the BlackBerry Enterprise Server. By default, wireless backup is active on the BlackBerry Enterprise Server. See the <i>BlackBerry Enterprise Server for Microsoft Exchange Feature and Technical Overview</i> for more information about the BlackBerry Enterprise Server.</p> <p>When the automatic wireless backup process runs on the BlackBerry Enterprise Server, the process saves BlackBerry Java® Application data with the user account settings and the other BlackBerry device data that backs up.</p>
Wireless (XML data)	<p>A BlackBerry® Java® Application uses XML APIs to generate and parse XML-formatted data to send and receive over a wireless connection.</p>
Desktop-based (BlackBerry® Desktop Manager Plug-in)	<p>A BlackBerry® Java® Application uses a USB connection to a computer to synchronize data with a desktop BlackBerry Java Application. This type of synchronization requires the use of the BlackBerry Desktop Synchronization APIs, the BlackBerry Desktop Manager, and a desktop BlackBerry Java Application that can read data from the BlackBerry device using the BlackBerry Desktop Manager Plug-Ins adapter. A BlackBerry device user must manually start the synchronization process by running the BlackBerry Desktop Manager Plug-in, which notifies the BlackBerry Java Application on the BlackBerry device to send the data to the desktop application.</p>
Desktop-based (USB protocols)	<p>A BlackBerry® Java® Application uses a USB connection to a computer and native USB protocols to synchronize data with a desktop application.</p>

Backing up and restoring data

Add support for backing up data over the wireless network

Task	Steps
Setup the BlackBerry® Enterprise Server™ to back up the BlackBerry Java® Application data using automatic wireless backup.	<ul style="list-style-type: none"> > Implement the OTASyncCapable and CollectionEventSource interfaces.
Activate the synchronization process when the BlackBerry® device starts.	<ul style="list-style-type: none"> > In the main method, create code that activates the synchronization process. <pre> public static void main(String[] args) { boolean startup = false; for (int i=0; i<args.length; ++i) { if (args[i].startsWith("init")) { startup = true; } } if (startup) { //enable application for synchronization on startup SerialSyncManager.getInstance().enableSynchronization(new RestaurantsSync()); } else { RestaurantsSync app = new RestaurantsSync(); app.enterEventDispatcher(); } } </pre> <p>The first time the BlackBerry device starts, the Alternate CLDC Application Entry Point project passes an argument to the BlackBerry Java® Application so that the BlackBerry Java Application registers only once.</p>
Create a project that acts as an alternate entry point to the main BlackBerry® Java® Application.	<p>MIDlet applications do not support this task.</p> <ol style="list-style-type: none"> 1. In the BlackBerry® Integrated Development Environment, create a project. 2. Right-click the project, and then click Properties. 3. Click the Application tab. 4. In the Project type drop-down list, click Alternate CLDC Application Entry Point. 5. In the Alternate entry point for drop-down list, click the project that starts the synchronization process. 6. In the Arguments passed to field, type init. Make sure the value you type in the Arguments passed to field matches the value in the startsWith argument in your BlackBerry Java Applications main method. 7. Select the Auto-run on startup option. 8. Select the System module option. 9. Click OK.

Task	Steps
Provide a BlackBerry® Java® Application with schema data for a SyncCollection.	<p>> In your implementation of the OTASyncCapable interface, implement the getSchema() method</p> <pre> public SyncCollectionSchema getSchema() { // returns our schema return _schema; } </pre>
Uniquely identify each record type in a SyncCollection.	<p>> Invoke the SyncCollectionSchema.setDefaultRecordType() method. The following example shows only one record type, so it uses the default record type:</p> <pre> private static final int DEFAULT_RECORD_TYPE = 1; _schema = new SyncCollectionSchema(); _schema.setDefaultRecordType(DEFAULT_RECORD_TYPE); </pre>
Uniquely identify each record in a SyncCollection.	<p>> Invoke the SyncCollectionSchema.setKeyFieldIds() method.</p> <pre> private static final int[] KEY_FIELD_IDS = new int[] {FIELDTAG_FIRST_NAME, FIELDTAG_LAST_NAME}; _schema.setKeyFieldIds(DEFAULT_RECORD_TYPE, KEY_FIELD_IDS); </pre>

Access a SyncCollection

Task	Steps
Retrieve an instance of the SyncCollection from the RunTimeStore.	<p>> To ensure the BlackBerry® Java® Application works with only one version of the SyncCollection, implement a static method that returns an instance of the SyncCollection.</p> <pre> static OTABackupRestoreContactCollection getInstance() { RuntimeStore rs = RuntimeStore.getRuntimeStore(); synchronized(rs) { OTABackupRestoreContactCollection collection = (OTABackupRestoreContactCollection)rs.get(AR_KEY); if(collection == null) { collection = new OTABackupRestoreContactCollection(); rs.put(AR_KEY, collection); } return collection; } } </pre>
Retrieve the SyncCollection from the PersistentStore.	<ol style="list-style-type: none"> To provide the BlackBerry® Java® Application with access to the newest SyncCollection data from the PersistentStore, invoke the PersistentStore.getPersistentObject() method using the ID of the SyncCollection. <pre> private PersistentObject _persist; // The persistable object for the contacts. private Vector _contacts; // The actual contacts. private static final long PERSISTENT_KEY = 0x266babf899b20b56L; _persist = PersistentStore.getPersistentObject(PERSISTENT_KEY); </pre> Store the returned data in a vector object. <pre> _contacts = (Vector)_persist.getContents(); </pre> Create a method to provide the BlackBerry Java Application with the newest SyncCollection data before a wireless data backup session begins. <pre> public void beginTransaction() { _persist = PersistentStore.getPersistentObject(PERSISTENT_KEY); _contacts = (Vector)_persist.getContents(); } </pre> Create code to manage the case where the SyncCollection you retrieve from the PersistentStore is empty. <pre> if(_contacts == null) { _contacts = new Vector(); _persist.setContents(_contacts); _persist.commit(); } </pre>

Notify the system when a SyncCollection changes

Task	Steps
Use a collection listener to notify the system when a SyncCollection changes.	<p>The system invokes <code>CollectionEventSource.addCollectionListener()</code> to create a <code>CollectionListener</code> for each <code>SyncCollection</code> the BlackBerry® Java® Application makes available for wireless backup.</p> <ol style="list-style-type: none"> 1. Create a private vector object to store the collection of <code>SyncCollection</code> listeners for the BlackBerry Java Application. <pre>private Vector _listeners; _listeners = new CloneableVector();</pre> 2. Implement the <code>CollectionEventSource.addCollectionListener()</code> method, making sure the method adds a <code>CollectionListener</code> to the vector. <pre>public void addCollectionListener(Object listener) { _listeners = ListenerUtilities.fastAddListener(_listeners, listener); }</pre>
Remove a collection listener.	<p>When a <code>CollectionListener</code> is no longer required, the system invokes <code>CollectionEventSource.removeCollectionListener</code>.</p> <ul style="list-style-type: none"> > Implement the <code>CollectionEventSource.removeCollectionListener()</code> method, using the <code>ListenerUtilities.removeListener()</code> method to remove a <code>CollectionListener</code> from the collection of <code>SyncCollection</code> listeners for the BlackBerry® Java® Application. <pre>public void removeCollectionListener(Object listener) { _listeners = ListenerUtilities.removeListener(_listeners, listener); }</pre>
Notify the system when an element is added to a SyncCollection.	<ul style="list-style-type: none"> > Invoke <code>CollectionListener.elementAdded()</code>: <pre>for(int i=0; i<_listeners.size(); i++) { CollectionListener cl = (CollectionListener)_listeners.elementAt(i); cl.elementAdded(this, object); } return true; }</pre>
Notify the system when an element is removed from a SyncCollection.	<ul style="list-style-type: none"> > Invoke <code>CollectionListener.elementRemoved()</code>.
Notify the system when an element in a SyncCollection is replaced.	<ul style="list-style-type: none"> > Invoke <code>CollectionListener.elementUpdated()</code>.

Using SyncObjects

Task	Steps
Retrieve SyncObjects from the SyncCollection.	<p>> Implement the <code>getSyncObjects()</code> method.</p> <pre> public SyncObject[] getSyncObjects() { //Retrieve the contact data. SyncObject[] contactArray = new SyncObject[_contacts.size()]; for (int i = _contacts.size() - 1; i >= 0; --i) { contactArray[i] = (SyncObject)_contacts.elementAt(i); } return contactArray; } </pre>
Access a specific SyncObject.	<p>> Implement the <code>getSyncObject()</code> method, using the <code>_uid</code> parameter to retrieve a specific SyncObject.</p> <pre> public SyncObject getSyncObject(int uid) { for (int i = _contacts.size() - 1; i >= 0; --i) { SyncObject so = (SyncObject)_contacts.elementAt(i); if (so.getUID() == uid) return so; } return null; } </pre>
Add a SyncObject to the SyncCollection.	<p>> Create a method that adds SyncObjects to the PersistentStore object.</p> <pre> public boolean addSyncObject(SyncObject object) { // Add a contact to the PersistentStore object. _contacts.addElement(object); } </pre>
Save a SyncCollection.	<p>Before a wireless backup session ends, save the newest SyncCollection data.</p> <p>> Invoke the <code>setContentts()</code> and <code>commit()</code> methods.</p> <pre> public void endTransaction() { _persist.setContentts(_contacts); _persist.commit(); } </pre>

See "Code sample: Using a SyncCollection to back up data over the wireless network" on page 91 for more information.

Add support for backing up data with the BlackBerry Desktop Software

Task	Steps
Let your BlackBerry® Java® Application maintain a collection of synchronized objects, producing and processing valid synchronization data when creating a SyncObject.	<p>Implement the <code>SyncCollection</code> and <code>SyncConverter</code> interfaces by the same class or by separate classes, depending on the design of the BlackBerry Java Application.</p> <ul style="list-style-type: none"> > Change the main class for the BlackBerry Java Application to implement the <code>SyncCollection</code> and <code>SyncConverter</code> interfaces. <pre>public class RestaurantsSync extends UiApplication implements RestaurantsSyncResource, SyncCollection, SyncConverter</pre>
Let persistable objects be synchronization objects.	<ul style="list-style-type: none"> > Modify a class that implements the <code>Persistable</code> interface to implement the <code>SyncObject</code> interface. <pre>private static final class RestaurantInfo implements Persistable, SyncObject {</pre>
Create a unique ID for a synchronization object.	<ul style="list-style-type: none"> > In the persistable class, create an instance variable for storing a unique ID for synchronization operations. <pre>private int _uid;</pre>
Let your main BlackBerry® Java® Application retrieve the unique ID of the synchronization object.	<ul style="list-style-type: none"> > In the persistable class, implement the <code>getUID()</code> method to return a unique ID for synchronization operations. <pre>public int getUID() { return _uid; }</pre>
Enable your main BlackBerry® Java® Application to create a synchronization object using a unique ID.	<ul style="list-style-type: none"> > In the persistable class, create a constructor that accepts a unique ID as a parameter and sets the <code>_uid</code> variable to this value. <pre>public RestaurantInfo(int uid) { _elements = new String[4]; for (int i = 0; i < _elements.length; ++i) { _elements[i] = ""; } _uid = uid; }</pre>

Activate synchronization when the BlackBerry device starts

Task	Steps
Activate synchronization when the BlackBerry® device starts.	<p>The first time the BlackBerry device starts, the Alternate CLDC Application Entry Point project passes an argument to the BlackBerry Java® Application so that the BlackBerry Java Application registers only once.</p> <ul style="list-style-type: none"> > In the main method of the BlackBerry Java Application, create code that activates the synchronization process. <pre> public static void main(String[] args) { boolean startup = false; for (int i=0; i<args.length; ++i) { if (args[i].startsWith("init")) { startup = true; } } if (startup) { //enable the BlackBerry Java Application for synchronization on startup SerialSyncManager.getInstance().enableSynchronization(new RestaurantsSync()); } else { RestaurantsSync app = new RestaurantsSync(); app.enterEventDispatcher(); } } </pre>
Create a project that acts as an alternate entry point to the main BlackBerry® Java® Application.	<p>If the BlackBerry Java Application is a MIDlet, arguments cannot pass to the BlackBerry Java Application when the BlackBerry device starts.</p> <ol style="list-style-type: none"> 1. In the BlackBerry® Integrated Development Environment, create a project. 2. Right-click the project, and then click Properties. 3. Click the Application tab. 4. In the Project type drop-down list, click Alternate CLDC Application Entry Point. 5. In the Alternate entry point for drop-down list, click the project that implements synchronization. 6. In the Arguments passed to field, type init. Make sure the value you type in the Arguments passed to field matches the value in the startsWith argument in the main method of the BlackBerry Java Application. 7. Select the Auto-run on startup option. 8. Select the System module option. 9. Click OK.

See “Code sample: Letting the BlackBerry Desktop Software to back up and restore BlackBerry Java Application data” on page 96 for more information.

Code samples

Code sample: Using a SyncCollection to back up data over the wireless network

Example: OTABackupRestoreContactCollection.java

```
/*
 * OTABackupRestoreContactCollection.java
 *
 * AUTO_COPYRIGHT_SUB_TAG
 */

package com.rim.samples.device.otabackuprestoredemo;

import java.io.*;
import java.util.*;
import net.rim.device.api.collection.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.synchronization.*;
import net.rim.device.api.util.*;
import net.rim.device.api.system.*;

/**
 * A collection enabled for OTA backup/restore. Basically a serially syncable collection
 * with few added interfaces.
 */
class OTABackupRestoreContactCollection implements SyncConverter, SyncCollection,
    OTASyncCapable, CollectionEventSource
{
    private static final long PERSISTENT_KEY = 0x266babf899b20b56L; //
com.rim.samples.device.otabackuprestoredemo.OTABackupRestoreContactCollection._persist
    private static final long AR_KEY = 0xef780e08b3a7cf07L; //
com.rim.samples.device.otabackuprestoredemo.OTABackupRestoreContactCollection

    private PersistentObject _persist; // the persistable object for the contacts
    private Vector _contacts; // the actual contacts
    private Vector _listeners; // listeners to generate events when contacts
are added
    private SyncCollectionSchema _schema; // lets us know about the data we are backing
up

    private static final int FIELDTAG_FIRST_NAME = 1;
    private static final int FIELDTAG_LAST_NAME = 2;
    private static final int FIELDTAG_EMAIL_ADDRESS = 3;

    private static final int DEFAULT_RECORD_TYPE = 1; // the id for the default (and
the only) record type
    private static final int[] KEY_FIELD_IDS = new int[] { // key fields - lets the server
know which fields uniquely define a record
        FIELDTAG_FIRST_NAME,
        FIELDTAG_LAST_NAME
    };
};
```

```

private OTABackupRestoreContactCollection()
{
    _persist = PersistentStore.getPersistentObject( PERSISTENT_KEY );
    _contacts = (Vector)_persist.getContents();

    if( _contacts == null )
    {
        _contacts = new Vector();
        _persist.setContents( _contacts );
        _persist.commit();
    }

    _listeners = new CloneableVector();

    // set up the schema for the collection
    _schema = new SyncCollectionSchema();
    _schema.setDefaultRecordType(DEFAULT_RECORD_TYPE);
    _schema.setKeyFieldIds(DEFAULT_RECORD_TYPE, KEY_FIELD_IDS);
}

static OTABackupRestoreContactCollection getInstance()
{
    RuntimeStore rs = RuntimeStore.getRuntimeStore();
    synchronized( rs )
    {
        OTABackupRestoreContactCollection collection =
(OTABackupRestoreContactCollection)rs.get( AR_KEY );
        if( collection == null )
        {
            collection = new OTABackupRestoreContactCollection();
            rs.put( AR_KEY, collection );
        }
        return collection;
    }
}

//SyncConverter methods-----
public boolean convert(SyncObject object, DataBuffer buffer, int version)
{
    if (version == getSyncVersion())
    {
        if (object instanceof ContactData)
        {
            String first = ((ContactData)object).getFirst();
            String last = ((ContactData)object).getLast();
            String email = ((ContactData)object).getEmail();

            //in compliance with desktop sync format
            buffer.writeShort(first.length()+1);
            buffer.writeByte(FIELDTAG_FIRST_NAME);
            buffer.write(first.getBytes());
            buffer.writeByte(0);
            buffer.writeShort(last.length()+1);
            buffer.writeByte(FIELDTAG_LAST_NAME);
            buffer.write(last.getBytes());
            buffer.writeByte(0);

```

```

        buffer.writeShort(email.length()+1);
        buffer.writeByte(FIELDTAG_EMAIL_ADDRESS);
        buffer.write(email.getBytes());
        buffer.writeByte(0);

        return true;
    }
}
return false;
}

public SyncObject convert(DataBuffer data, int version, int UID)
{
    try {
        ContactData contact = new ContactData(UID);
        while(data.available() > 0)
        {
            int length = data.readShort();
            byte[] bytes = new byte[length];
            switch (data.readByte())
            {
                case FIELDTAG_FIRST_NAME:
                    data.readFully(bytes);
                    //trim null-terminator
                    contact.setFirst(new String(bytes).trim());
                    break;
                case FIELDTAG_LAST_NAME:
                    data.readFully(bytes);
                    contact.setLast(new String(bytes).trim());
                    break;
                case FIELDTAG_EMAIL_ADDRESS:
                    data.readFully(bytes);
                    contact.setEmail(new String(bytes).trim());
                    break;
                default:
                    data.readFully(bytes);
                    //other fields not supported
                    break;
            }
        }
        return contact;
    }
    catch (EOFException e)
    {
        System.err.println(e.toString());
    }
    return null;
}

//SyncCollection methods-----
public boolean addSyncObject(SyncObject object)
{
    // add a contact to the persistent store
    _contacts.addElement(object);
    _persist.setContents( _contacts );
    _persist.commit();
}

```

```

changed // we want to let any collection listeners we have that the collection has been
        for( int i=0; i<_listeners.size(); i++ )
        {
            CollectionListener cl = (CollectionListener)_listeners.elementAt( i );
            cl.elementAdded( this, object );
        }
        return true;
    }

    public boolean updateSyncObject(SyncObject oldObject, SyncObject newObject)
    {
        return false; //na - this method would look much the same as addSyncObject
    }

    public boolean removeSyncObject(SyncObject object)
    {
        return false; //na - this method would look much the same as addSyncObject
    }

    public boolean removeAllSyncObjects()
    {
        return false; //na
    }

    public SyncObject[] getSyncObjects()
    {
        SyncObject[] contactArray = new SyncObject[_contacts.size()];
        for (int i = _contacts.size() - 1; i >= 0; --i)
        {
            contactArray[i] = (SyncObject)_contacts.elementAt(i);
        }
        return contactArray;
    }

    public SyncObject getSyncObject(int uid)
    {
        for (int i = _contacts.size() - 1; i >= 0; --i)
        {
            SyncObject so = (SyncObject)_contacts.elementAt(i);
            if ( so.getUID() == uid ) return so;
        }
        return null;
    }

    public boolean isSyncObjectDirty(SyncObject object)
    {
        return false; //na
    }

    public void setSyncObjectDirty(SyncObject object)
    {
        //na
    }

    public void clearSyncObjectDirty(SyncObject object)
    {
        //na
    }

```

```

    }

    public int getSyncObjectCount()
    {
        _persist = PersistentStore.getPersistentObject(PERSISTENT_KEY);
        _contacts = (Vector)_persist.getContents();
        return _contacts.size();
    }

    public int getSyncVersion()
    {
        return 1;
    }

    public String getSyncName()
    {
        return "OTABackupRestoreContacts";
    }

    public String getSyncName(Locale locale)
    {
        return null;
    }

    public SyncConverter getSyncConverter()
    {
        return this;
    }

    public void beginTransaction()
    {
        _persist = PersistentStore.getPersistentObject(PERSISTENT_KEY);
        _contacts = (Vector)_persist.getContents();
    }

    public void endTransaction()
    {
        _persist.setContents(_contacts);
        _persist.commit();
    }

    //OTASyncCapable methods -----
    public SyncCollectionSchema getSchema()
    {
        // returns our schema
        return _schema;
    }

    //CollectionEventSource methods -----
    public void addCollectionListener(Object listener)
    {
        _listeners = ListenerUtilities.fastAddListener( _listeners, listener );
    }

    public void removeCollectionListener(Object listener)
    {
        _listeners = ListenerUtilities.removeListener( _listeners, listener );
    }

```

```

    public int size()
    {
        return _contacts.size();
    }

    public ContactData contactAt( int index )
    {
        return (ContactData)_contacts.elementAt( index );
    }
}

```

Code sample: Letting the BlackBerry Desktop Software to back up and restore BlackBerry Java Application data

Example: RestaurantsSync.java

```

/**
 * RestaurantsSync.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.restaurantssync;

import java.io.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;
import java.util.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.synchronization.*;
import com.rim.samples.docs.resource.*;

public class RestaurantsSync extends UiApplication implements RestaurantsSyncResource,
    SyncCollection, SyncConverter {

    private static final long KEY = 0xdec6a67096f833cL;

    private AutoTextEditField namefield;
    private AutoTextEditField addressfield;
    private EditField phonefield;
    private EditField specialtyfield;

    private static PersistentObject store;
    private static Vector _data;
    private static ResourceBundle _resources;
    private static final int FIELDTAG_NAME = 1;
    private static final int FIELDTAG_PHONE = 2;
    private static final int FIELDTAG_ADDRESS = 3;
    private static final int FIELDTAG_SPECIALTY = 4;

    private static RestaurantsSync _instance;

```

```

private MenuItem saveItem = new MenuItem(_resources, MENUITEM_SAVE, 110, 10) {
    public void run() {
        RestaurantInfo info = new RestaurantInfo();
        info.setElement(RestaurantInfo.NAME, namefield.getText());
        info.setElement(RestaurantInfo.ADDRESS, addressfield.getText());
        info.setElement(RestaurantInfo.PHONE, phonefield.getText());
        info.setElement(RestaurantInfo.SPECIALTY, specialtyfield.getText());
        _data.addElement(info);

        synchronized(store) {
            store.setContents(_data);
            store.commit();
        }
        Dialog.inform(_resources.getString(APP_SUCCESS));
        namefield.setText(null);
        addressfield.setText(null);
        phonefield.setText("");
        specialtyfield.setText("");
    }
};

private MenuItem getItem = new MenuItem(_resources, MENUITEM_GET, 110, 11) {
    public void run() {
        synchronized(store) {
            _data = (Vector)store.getContents();
            if (!_data.isEmpty()) {
                RestaurantInfo info = (RestaurantInfo)_data.lastElement();
                namefield.setText(info.getElement(RestaurantInfo.NAME));
                addressfield.setText(info.getElement(RestaurantInfo.ADDRESS));
                phonefield.setText(info.getElement(RestaurantInfo.PHONE));
                specialtyfield.setText(info.getElement(
                    RestaurantInfo.SPECIALTY));
            }
        }
    }
};

static {
    _resources = ResourceBundle.getBundle(RestaurantsSyncResource.BUNDLE_ID,
        RestaurantsSyncResource.BUNDLE_NAME);
    store = PersistentStore.getPersistentObject(KEY);
    synchronized (store) {
        _data = (Vector)store.getContents();
        if ( _data == null ) {
            _data = new Vector();
            store.setContents( _data );
            store.commit();
        }
    }
}

public static void main(String[] args) {
    boolean startup = false;
    for (int i=0; i<args.length; ++i) {
        if (args[i].startsWith("init")) {
            startup = true;
        }
    }
}

```

```

    }
}

if (startup) {
    // Enable application for synchronization on startup.
    SyncManager.getInstance().enableSynchronization(
        RestaurantsSync.getInstance());
} else {
    RestaurantsSync app = new RestaurantsSync();
    app.enterEventDispatcher();
}

}

public static RestaurantsSync getInstance() {
    if (_instance == null) {
        _instance = new RestaurantsSync();
    }
    return _instance;
}

private static final class RestaurantInfo implements Persistable, SyncObject {
    private String[] _elements; // Data.
    public static final int NAME = 0;
    public static final int ADDRESS = 1;
    public static final int PHONE = 2;
    public static final int SPECIALTY = 3;
    private int _uid;

    public int getUID() {
        return _uid;
    }

    public RestaurantInfo() {
        _elements = new String[4];
        for (int i = 0; i < _elements.length; ++i) {
            _elements[i] = "";
        }
    }

    public RestaurantInfo(int uid) {
        _elements = new String[4];
        for (int i = 0; i < _elements.length; ++i) {
            _elements[i] = "";
        }
        _uid = uid;
    }

    public String getElement(int id) {
        return _elements[id];
    }

    public void setElement(int id, String value) {
        _elements[id] = value;
    }
}

// SyncConverter methods.
public SyncObject convert(DataBuffer data, int version, int UID) {
    try {

```

```

        RestaurantInfo info = new RestaurantInfo(UUID);
        while(data.available() > 0) {
            int length = data.readShort();
            byte[] bytes = new byte[length];
            switch (data.readByte()) {
                case FIELDTAG_NAME:
                    data.readFully(bytes);
                    //trim null-terminator
                    info.setElement(RestaurantInfo.NAME,
                        new String(bytes).trim());
                    break;
                case FIELDTAG_PHONE:
                    data.readFully(bytes);
                    info.setElement(RestaurantInfo.PHONE,
                        new String(bytes).trim());
                    break;
                case FIELDTAG_ADDRESS:
                    data.readFully(bytes);
                    info.setElement(RestaurantInfo.ADDRESS,
                        new String(bytes).trim());
                    break;
                case FIELDTAG_SPECIALTY:
                    data.readFully(bytes);
                    info.setElement(RestaurantInfo.SPECIALTY,
                        new String(bytes).trim());
                    break;
                default:
                    data.readFully(bytes);
                    break;
            }
        }
        return info;
    } catch (EOFException e) {
        System.err.println(e.toString());
    }
    return null;
}

public boolean convert(SyncObject object, DataBuffer buffer, int version) {
    if (version == getSyncVersion()) {
        if (object instanceof RestaurantInfo )
        {
            String name = ((RestaurantInfo)object).getElement(
                RestaurantInfo.NAME);
            String phone = ((RestaurantInfo)object).getElement(
                RestaurantInfo.PHONE);
            String address = ((RestaurantInfo)object).getElement(
                RestaurantInfo.ADDRESS);
            String specialty = ((RestaurantInfo)object).getElement(
                RestaurantInfo.SPECIALTY);
            buffer.writeShort(name.length()+1);
            buffer.writeByte(FIELDTAG_NAME);
            buffer.write(name.getBytes());
            buffer.writeByte(0);
            buffer.writeShort(phone.length()+1);
            buffer.writeByte(FIELDTAG_PHONE);
            buffer.write(phone.getBytes());
            buffer.writeByte(0);

```

```

        buffer.writeShort(address.length()+1);
        buffer.writeByte(FIELDTAG_ADDRESS);
        buffer.write(address.getBytes());
        buffer.writeByte(0);
        buffer.writeShort(specialty.length()+1);
        buffer.writeByte(FIELDTAG_SPECIALTY);
        buffer.write(specialty.getBytes());
        buffer.writeByte(0);
        return true;
    }
}
return false;
}

public void beginTransaction() {
    store = PersistentStore.getPersistentObject(KEY);
    _data = (Vector)store.getContents();
}

public void endTransaction() {
    store.setContents(_data);
    store.commit();
}

public SyncConverter getSyncConverter() {
    return this;
}

public String getSyncName() {
    return "Restaurant Synchronization Demo";
}

public String getSyncName(Locale locale) {
    return getSyncName();
}

public int getSyncObjectCount() {
    store = PersistentStore.getPersistentObject(KEY);
    _data = (Vector)store.getContents();
    return _data.size();
}

public SyncObject[] getSyncObjects() {
    SyncObject[] array = new SyncObject[_data.size()];
    for (int i = _data.size() - 1; i >= 0; --i) {
        array[i] = (SyncObject)_data.elementAt(i);
    }
    return array;
}

public SyncObject getSyncObject(int uid) {
    for (int i = _data.size() - 1; i >= 0; --i) {
        SyncObject so = (SyncObject)_data.elementAt(i);
        if (so.getUID() == uid ) return so;
    }
    return null;
}

```

```

public int getSyncVersion() {
    return 1;
}

public boolean addSyncObject(SyncObject object) {
    _data.addElement(object);
    return true;
}

public boolean removeAllSyncObjects() {
    _data.removeAllElements();
    return true;
}

public void clearSyncObjectDirty(SyncObject object) {
    // Not applicable.
}

public boolean isSyncObjectDirty(SyncObject object) {
    return false;
}

public boolean removeSyncObject(SyncObject object) {
    return false;
}

public void setSyncObjectDirty(SyncObject object) {
}

public boolean updateSyncObject(SyncObject oldObject, SyncObject newObject) {
    return false;
}

public RestaurantsSync() {
    MainScreen mainScreen = new RestaurantsMainScreen();
    mainScreen.setTitle(new LabelField( _resources.getString(APPLICATION_TITLE)));
    namefield = new AutoTextEditField( _resources.getString(FIELD_NAME), "");
    addressfield = new AutoTextEditField( _resources.getString(FIELD_ADDRESS), "");
    phonefield = new EditField(
        "", Integer.MAX_VALUE, BasicEditField.FILTER_PHONE);
    specialtyfield = new EditField(
        _resources.getString(FIELD_SPECIALTY), "",
        Integer.MAX_VALUE, BasicEditField.FILTER_DEFAULT);
    mainScreen.add(namefield);
    mainScreen.add(addressfield);
    mainScreen.add(phonefield);
    mainScreen.add(specialtyfield);
    pushScreen(mainScreen);
}

private final class RestaurantsMainScreen extends MainScreen
{
    protected void makeMenu( Menu menu, int instance ) {
        menu.add(saveItem);
        menu.add(getItem);
        super.makeMenu(menu, instance);
    }

    public void close() {

```

```
        Dialog.alert(_resources.getString(APP_EXIT));  
        super.close();  
    }  
}
```

Implementing security and trust services

Connecting to an application on a SIM card

Connecting to an application on a SIM card

For BlackBerry® devices that operate on EDGE networks, you can create BlackBerry Java® Applications that use the APDU and JCRMI APIs defined by JSR 177, to call the methods of an application on a SIM card. You must use the BlackBerry® Java® Development Environment Version 4.2.1 or greater to create a BlackBerry Java Application that uses the APDU and JCRMI APIs.

The APDU API enables BlackBerry Java Applications on a BlackBerry device to use the APDU protocol to communicate with an application on a SIM card.

The JCRMI API enables BlackBerry Java Applications on a BlackBerry device to use the Java Card Remote Method Invocation protocol to communicate with a BlackBerry Java Application on a SIM card.

For more information on creating BlackBerry Java Applications that support JSR177, see <http://jcp.org/en/home/index>.

PIN ID restrictions for JSR177 on a BlackBerry device

You can load up to eight applications onto a SIM card. The BlackBerry Java Applications are classified as applications one through eight. You can associate two PIN IDs with each BlackBerry Java Application. A BlackBerry Java Application only requires PIN IDs for PIN related functions in JSR177 such as EnterPin, UnblockPin, EnablePin, and ChangePin.

The 3GPP USIM application is listed as application one on the SIM card and uses PIN IDs 0x01 and 0x81.

BlackBerry Java Applications that implement JRS177 can only use the following PIN ID ranges: 0x02 to 0x08, and 0x82 to 0x88. Note that PIN1 for application i is defined as 0x0*i*, where $1 \leq i \leq 8$, and PIN2 for the same application is defined as 0x8*i*.

Use JCRMI to communicate with an object on a SIM card

1. Create an interface that extends the `java.rmi.Remote` interface and includes the methods the BlackBerry® Java® Application will call on the remote object.

```
public interface Wallet extends Remote {
    public short getBalance() throws RemoteException, UserException;
    public void debit(short m) throws RemoteException, UserException;
    public void credit(short m) throws RemoteException, UserException;
    public byte[] getAccountNumber() throws RemoteException, UserException;}

```

2. Create a stub class that extends the `javax.microedition.jcrmi.RemoteStub` class and implements the `java.rmi.Remote` interface.

```
public class Wallet_Stub extends RemoteStub implements Remote, Wallet {
    public short getBalance() throws RemoteException, UserException {
        try {
            Object $result= ref.invoke("getBalance()S", null);
            return ((java.lang.Short) $result).shortValue();
        } catch (java.lang.RuntimeException e) {
            throw e;
        } catch (java.rmi.RemoteException e) {
            throw e;
        } catch (javacard.framework.UserException e) {
            throw e;
        } catch (java.lang.Exception e) {
            throw new java.rmi.RemoteException("undeclared checked exception", e);
        }
    }

    public void debit(short m) throws RemoteException, UserException {
        try {
            ref.invoke("debit(S)V", new java.lang.Object[] {new java.lang.Short(m)});
        } catch (java.lang.RuntimeException e) {
            throw e;
        } catch (java.rmi.RemoteException e) {
            throw e;
        } catch (javacard.framework.UserException e) {
            throw e;
        } catch (java.lang.Exception e) {
            throw new java.rmi.RemoteException("undeclared checked exception", e);
        }
    }
}

```

```

public void credit(short m) throws RemoteException, UserException {
    try {
        ref.invoke("credit(S)V", new java.lang.Object[] {new java.lang.Short(m)});
    } catch (java.lang.RuntimeException e) {
        throw e;
    } catch (java.rmi.RemoteException e) {
        throw e;
    } catch (javacard.framework.UserException e) {
        throw e;
    } catch (java.lang.Exception e) {
        throw new java.rmi.RemoteException("undeclared checked exception", e);
    }
}

```

```

public byte[] getAccountNumber() throws RemoteException, UserException {
    try {
        Object $result=ref.invoke("getAccountNumber()[B", null);
        return (byte[])$result;
    } catch (java.lang.RuntimeException e) {
        throw e;
    } catch (java.rmi.RemoteException e) {
        throw e;
    } catch (javacard.framework.UserException e) {
        throw e;
    } catch (java.lang.Exception e) {
        throw new java.rmi.RemoteException("undeclared checked exception", e);
    }
}

```

```

}

```

3. Create BlackBerry Java Application code that opens a connection to and calls methods on the remote object.

```

JavaCardRMICConnection connection = null;
connection = (JavaCardRMICConnection)
Connector.open("jcrmi:0;AID=A0.0.0.67.4.7.1F.3.2C.3");
if(connection==null){
    fail("Null connection returned");
}

```

Use APDU to communicate with an object on a SIM card

You can create a BlackBerry® Java® Application that uses the APDU protocol to send and receive message information between a BlackBerry Java Application and an application on a SIM card. See the *API Reference* for more information on using the `APDUConnection` interface.

Task	Steps
Open an APDU connection.	<ol style="list-style-type: none"> 1. Create a locator string that begins with <code>apdu</code> and specifies the slot number and the card application identifier. <pre>String testURL = "apdu:0;target=A0.0.0.67.4.7.1F.3.2C.3";</pre> 2. Invoke <code>Connector.open()</code> using the locator string and casting the returned object as an <code>APDUConnection</code> object. <pre>APDUConnection acn = null; acn = (APDUConnection)Connector.open(testURL);</pre>
Send and receive APDU messages.	<ol style="list-style-type: none"> 1. Create a byte array containing a command APDU. <pre>byte[] command = {(byte)0x00, (byte)0x24, (byte)0x04, (byte)0x64};</pre> 2. Invoke <code>exchangeAPDU()</code> using the byte array command, storing the return value in a byte array. <pre>byte[] responseAPDU = acn.exchangeAPDU(command);</pre>
Close the APDU connection.	<ul style="list-style-type: none"> > Invoke <code>APDUConnection.close()</code>. <pre>acn.close();</pre>

Managing memory

Invoking garbage collection
Reduce the number of objects
Managing low memory

Invoking garbage collection

See the *Garbage Collection in the BlackBerry® Java® Development Environment* whitepaper and the *BlackBerry Java Development Environment Fundamentals Guide* for more information about garbage collection operations.

Reduce the number of objects

To use the BlackBerry® Integrated Development Environment to identify unnecessary objects, complete the following steps:

1. Open the BlackBerry IDE.
2. Place two breakpoints in the code surrounding an area of high object creation.
3. Run the BlackBerry Java® Application to the first breakpoint.
4. Open the **Objects** window and click **Snapshot**.
5. Run the BlackBerry Java Application to the second breakpoint.
6. Open the **Objects** window.
7. Click **Compare to Snapshot**.
8. View multiple snapshots in the **Objects** window.
9. Determine which objects can be removed.

Managing low memory

LMM triggers

The following conditions can cause the LMM to free memory resources:

Condition	Description
Available flash memory falls below acceptable thresholds.	The free flash memory threshold depends on the amount of free RAM in the system. The free flash memory threshold varies between 400 KB and 800 KB.

Condition	Description
Low number of persistent object handles on a BlackBerry® device..	The number of persistent object handles falls below 1000.
Low number of object handles on a BlackBerry® device	The number of object handles falls below 1000.

Use the LMM

Task	Steps
Register your BlackBerry® Java® Application with the LMM.	<ol style="list-style-type: none"> 1. In the BlackBerry Java Application, implement the <code>LowMemoryListener</code> interface. 2. Enable the BlackBerry Java Application to register the <code>LowMemoryListener</code> with the LMM when the BlackBerry Java Application starts for the first time. Register the listener only once.
Manage events that the <code>LowMemoryListener</code> receives.	<ul style="list-style-type: none"> > Implement the <code>freeStaleObject(int)</code> method of the <code>LowMemoryListener</code> interface. The implementation of <code>freeStaleObject()</code> method should return true if persistent data is released, or return false otherwise.
Manage low priority events.	<p>The LMM seldom specifies a priority higher than low priority.</p> <ul style="list-style-type: none"> > In an implementation of <code>freeStaleObject()</code>, enable the BlackBerry® Java® Application to release transitory variables and any variables that are currently not necessary for complete functionality, such as cached data.
Manage medium priority events.	<ul style="list-style-type: none"> > In an implementation of <code>freeStaleObject()</code>, enable the BlackBerry® Java® Application to remove stale data, such as very old email messages or old calendar appointments.
Manage high priority events.	<ul style="list-style-type: none"> > In an implementation of <code>freeStaleObject()</code>, enable the BlackBerry® Java® Application to remove objects in the BlackBerry Java Application on a Least Recently Used basis, removing all stale objects.
Free resources manually.	<p>The BlackBerry® device system invokes the implementation of <code>freeStaleObject(int)</code> when device memory is low. To manually free resources, perform the following actions:</p> <ul style="list-style-type: none"> > Invoke <code>freeStaleObject(int)</code> from the BlackBerry Java® Application.

Free persistent objects

Task	Steps
Remove references to the object.	<ol style="list-style-type: none"> 1. In the BlackBerry® Java® Application, remove references to the persistent object. 2. Delete the object from its data structure.
Inform the BlackBerry® JVM that it can now remove the object.	<ul style="list-style-type: none"> > In an implementation of <code>freeStaleObject()</code>, invoke <code>LowMemoryManager.markAsRecoverable()</code>.
Commit changes to data collections.	<ul style="list-style-type: none"> > In an implementation of <code>freeStaleObject()</code>, invoke <code>PersistentObject.commit()</code>.

Creating connections

- Fetching data using HTTP or TCP sockets
- Working with network information
- Datagram connections
- Using port connections
- Code samples

Fetching data using HTTP or TCP sockets

BlackBerry® Java® Applications for BlackBerry devices can use standard HTTP, HTTPS, and TCP socket protocols to establish connections over the wireless network. When establishing the connection over the cellular network, a BlackBerry Java Application can use one of two wireless gateways to proxy the connection to the Internet or the corporate intranet. You can design your BlackBerry Java Application to rely on the default gateway that is available to the BlackBerry device user, or you can customize your code to explicitly select a preferred gateway.

Working with network information

Determine the name of the wireless network that the BlackBerry device is registered with

> Invoke `RadioInfo.getCurrentNetworkName()`.

The BlackBerry device must be registered with a wireless network for this method to work.

```
String networkName = RadioInfo.getCurrentNetworkName();  
System.out.println ("Network Name: " + networkName );
```

Verify that the BlackBerry device is in network coverage

> Use the `CoverageInfo` class and `CoverageStatusListener` interface of the `net.rim.device.api.system` package to make sure that the BlackBerry® device is in network coverage.

Explicitly selecting a gateway

Set up your BlackBerry® Java® Application to use the preferred gateway for a connection and to use the default gateway only when the preferred gateway is not available.

Using the BlackBerry Enterprise Server as an intranet gateway

Enterprise customers host the BlackBerry® Enterprise Server behind their corporate firewall to enable access from BlackBerry devices to the corporate intranet. The BlackBerry Mobile Data System™ component of the BlackBerry Enterprise Server includes the BlackBerry MDS™ Services, which provides an HTTP and TCP/IP proxy service to let third-party BlackBerry Java® Applications use it as a secure gateway for managing HTTP and TCP/IP connections to the intranet. When you use the BlackBerry Enterprise Server as an intranet gateway, all traffic between your BlackBerry Java Application and the BlackBerry Enterprise Server is automatically encrypted using AES or triple DES encryption. Because the BlackBerry Enterprise Server resides behind the corporate firewall and provides inherent data encryption, BlackBerry Java Applications can communicate with application servers and web servers that reside on the corporate intranet.

If your BlackBerry Java Application connects to the Internet rather than to the corporate intranet, you might be able to use the BlackBerry Enterprise Server that belongs to the customer as a gateway as well. In this case, network requests travel behind the corporate firewall to the BlackBerry Enterprise Server, which makes the network request to the Internet through the corporate firewall. However, enterprise customers can set an IT policy to enforce that the BlackBerry Enterprise Server is the gateway for all wireless network traffic, including traffic destined for the Internet.

If your BlackBerry Java Application connects to the Internet, and you are targeting non-enterprise customers, you can also use either the BlackBerry Internet Service or the Internet gateway of the wireless server provider to manage connections.

Using the wireless service provider's Internet gateway

BlackBerry® Java® Applications for BlackBerry devices can connect to the Internet using the Internet gateway that the wireless service provider provides. Most wireless service providers provide their own Internet gateway that offers direct TCP/IP connectivity to the Internet. Some operators also provide a WAP gateway that lets HTTP connections occur over the WAP protocol. BlackBerry Java Applications for BlackBerry devices can use either of these gateways to establish connections to the Internet. If you write your BlackBerry Java Application for BlackBerry device users who are on a specific wireless network, this approach can often yield good results. However, if you write your BlackBerry Java Application for BlackBerry device users on a variety of wireless networks, testing your BlackBerry Java Application against the different Internet gateways and achieving a consistent and reliable experience can be challenging. In these scenarios, you may find it useful to use the BlackBerry Internet Service, and use the wireless service provider's Internet gateway as a default connection type if the BlackBerry Internet Service is not available.

In the Technical Knowledge Center on the BlackBerry Developer Zone, see the whitepaper *Managing Wireless Data Transport in the BlackBerry Solution v4.0 Part 1: Understanding TCP and HTTP transport options for Java applications for BlackBerry* for more information on managing wireless connectivity and how to effectively use each of the gateways.

Use HTTP connections

Task	Steps
Before opening an HTTP connection, verify that the BlackBerry® device is in network coverage.	> Use the <code>CoverageInfo</code> class and <code>CoverageStatusListener</code> interface of the <code>net.rim.device.api.system</code> package to make sure that the BlackBerry device is in network coverage.
Open an HTTP connection.	<ol style="list-style-type: none"> 1. Invoke <code>Connector.open()</code>, specifying http as the protocol. 2. Cast the returned object as an <code>HttpConnection</code> or a <code>StreamConnection</code> object. <pre>HttpConnection conn = null; String URL = "http://www.myServer.com/myContent"; conn = (HttpConnection)Connector.open(URL);</pre>
Set the HTTP request method (GET or POST).	> Invoke <code>HttpConnection.setRequestMethod()</code> . <pre>Conn.setRequestMethod(HttpConnection.POST);</pre>
Set header fields.	> Invoke <code>setRequestProperty()</code> on the <code>HttpConnection</code> . <pre>conn.setRequestProperty("User-Agent", "BlackBerry/3.2.1");</pre>
Retrieve header fields.	> Invoke <code>getRequestProperty()</code> on the <code>HttpConnection</code> . <pre>String lang = conn.getRequestProperty("Content-Language");</pre>
Send and receive data.	> Invoke <code>openInputStream()</code> and <code>openOutputStream()</code> on the <code>HttpConnection</code> . <pre>InputStream in = conn.openInputStream(); OutputStream out = conn.openOutputStream();</pre>

See "Code sample: Using an HTTP connection to retrieve data" on page 120 for more information.

Use HTTP authentication

Task	Steps
Before opening an HTTP connection, verify that the BlackBerry® device is in network coverage.	> Use the <code>CoverageInfo</code> class and <code>CoverageStatusListener</code> interface of the <code>net.rim.device.api.system</code> package to make sure that the BlackBerry device is in network coverage.
Open an HTTP connection.	<ol style="list-style-type: none"> 1. Invoke <code>Connector.open()</code>, using the HTTP location of the protected resource. 2. Cast and store the returned object as a <code>StreamConnection</code>. <pre>StreamConnection s = (StreamConnection)Connector.open("http://mysite.com/myProtectedFile.txt");</pre> 3. Cast and store the <code>StreamConnection</code> object as an <code>HTTPConnection</code> object. <pre>HttpConnection httpConn = (HttpConnection)s;</pre>
Determine the status of the HTTP connection.	> Invoke <code>HttpConnection.getResponseCode()</code> . <pre>int status = httpConn.getResponseCode();</pre>
Retrieve login information from a BlackBerry® device user.	<ol style="list-style-type: none"> 1. Create code that manages an unauthorized HTTP connection attempt. <pre>int status = httpConn.getResponseCode(); switch (status) case (HttpConnection.HTTP_UNAUTHORIZED);</pre> 2. Create a <code>run()</code> method and within it implement a dialog object to ask the BlackBerry device user for login information. <pre>UiApplication.getUiApplication().invokeAndWait(new Runnable() { public void run() { dialogResponse = Dialog.ask; (Dialog.D_YES_NO,"Unauthorized Access:\n Do you wish to log in?"); } }</pre>
Process the response of the BlackBerry® device user.	<ol style="list-style-type: none"> 1. Create code that manages a Yes dialog response. 2. Retrieve the login information and close the current connection. <pre>if (dialogResponse == Dialog.YES) {String login = "username:password"; //Close the connection. s.close();</pre> 3. Encode the login information. <pre>byte[] encoded = Base64OutputStream.encode(login.getBytes(), 0, login.length(), false, false);</pre>
Use the BlackBerry® device user login information to access the protected resource.	> Open a new <code>HTTPConnection</code> and add the authorization header by invoking <code>HttpConnection.setRequestProperty()</code> using the encoded login information. <pre>s = (StreamConnection)Connector.open("http://mysite.com/myProtectedFile.txt "); httpConn = (HttpConnection)s; httpConn.setRequestProperty("Authorization", "Basic " + new String(encoded));</pre>

Code fragment: Using HTTP authentication to connect to a protected internet resource

Example: Using HTTP authentication to connect to a protected Internet resource

```

HttpConnection httpConn = null;
StreamConnection s = null;
boolean keepGoing = true;
int dialogResponse;

try
{
s = (StreamConnection)Connector.open("http://mysite.com/myProtectedFile.txt");
httpConn = (HttpConnection)s;

while(keepGoing)
{
    int status = httpConn.getResponseCode();
    switch (status)
    {
        case (HttpConnection.HTTP_OK):
            //Connection is 200 OK.
            //Download and process data.
            keepGoing = false;
            break;

        case (HttpConnection.HTTP_UNAUTHORIZED):
            //Connection is 401 Unauthorized.
            //A login and password is required.

            //Retrieve the login information from somewhere.
            //You could prompt the user for this information or
            //retrieve this from elsewhere if it is saved within
            //your application.
            //Login information is hard coded here for brevity, but
            //we ask the user if they want to log-in.

            UiApplication.getUiApplication().invokeAndWait(new Runnable()

            {
                public void run()
                {
                    dialogResponse = Dialog.ask
                    (Dialog.D_YES_NO,"Unauthorized Access:\n Do you wish to log in?");
                }
            });

            if (dialogResponse == Dialog.YES)
            {
                String login = "username:password";

                //Close the connection.
            }
        }
    }
}

```

```
s.close();

//Encode the login information in Base64 format.
byte[] encoded = Base64OutputStream.encode(login.getBytes(), 0,
login.length(), false, false);

//Open a new connection.
s = (StreamConnection)Connector.open("http://mysite.com/myProtectedFile.txt
");
httpConn = (HttpConnection)s;

//Add the authorized header.
httpConn.setRequestProperty("Authorization", "Basic " + new String(encoded));
}

else
{
//Handle failed connection.
keepGoing = false;
}

break;

default:
//The connection failed for some other reason.
//Handle failed connection.
keepGoing = false;
break;
}

}
//Close the connection.
s.close();
}
catch (IOException e)
{
//Handle the exception.
}
}
```

Use HTTPS connections

Task	Steps
Before opening an HTTPS connection, verify that the BlackBerry® device is within network coverage.	> Use the <code>CoverageInfo</code> class and <code>CoverageStatusListener</code> interface of the <code>net.rim.device.api.system</code> package to make sure that the BlackBerry device is in network coverage.
Open an HTTPS connection.	<ol style="list-style-type: none"> 1. Invoke <code>Connector.open()</code>, specifying HTTPS as the protocol. 2. Cast the returned object as an <code>HttpsConnection</code> object. <pre>HttpsConnection stream = (HttpsConnection)Connector.open("https://host:443/");</pre>
Specify the connection mode.	<p>If your BlackBerry device is associated with a BlackBerry Enterprise Server and uses an HTTPS proxy server that requires authentication, you will not be able to use end-to-end TLS.</p> <p>> To open an HTTPS connection in end-to-end mode, add one of the following parameters to the connection string that passes to <code>Connector.open()</code>:</p> <ul style="list-style-type: none"> • Specify that an end-to-end HTTPS connection must be used from the BlackBerry® device to the target server: <code>EndToEndRequired</code>. • Specify that an end-to-end HTTPS connection should be used from the BlackBerry device to the target server. If the BlackBerry device does not support end-to-end TLS, and the BlackBerry device user permits proxy TLS connections, then a proxy connection is used: <code>EndToEndDesired</code>. <pre>HttpsConnection stream = (HttpsConnection)Connector.open("https://host:443/;EndToEndDesired");</pre>

Use socket connections

Although you can implement HTTP over a socket connection, you should use an HTTP connection for the following reasons:

- Socket connections do not support BlackBerry® Mobile Data System™ features, such as push.
- BlackBerry® Java® Applications that use socket connections typically require significantly more bandwidth than BlackBerry Java Applications that use HTTP connections



Note:In the 'Open a Socket connection' tasks that follow, the *deviceside* parameter will specify whether or not the connection uses BlackBerry® MDS Services [(*deviceside=false*)] or direct TCP [(*deviceside=true*)].

If you do not specify the optional *deviceside* parameter, the following results occur:

- The connection uses direct TCP by default for any BlackBerry on the iDEN network (BlackBerry 6510 Wireless Handheld, BlackBerry 7510, BlackBerry 7520, BlackBerry 7100i).
- On all other BlackBerry devices, BlackBerry MDS Services is used by default. If BlackBerry MDS Services is not available, the BlackBerry device uses direct TCP.

Task	Steps
Before opening a socket connection, verify that the BlackBerry® device is in network coverage.	> Use the <code>CoverageInfo</code> class and <code>CoverageStatusListener</code> interface of the <code>net.rim.device.api.system</code> package to make sure that the BlackBerry device is in network coverage.

Task	Steps
Open a socket connection using BlackBerry® MDS Services.	<p>> Invoke <code>Connector.open()</code>, specifying <code>socket</code> as the protocol and appending the <code>deviceside=false</code> parameter to the end of the URL. BlackBerry Java® Applications must input their local machine IP explicitly because <code>localhost</code> is not supported.</p> <pre>private static String URL = "socket:// local_machine_IP:4444;deviceside=false"; StreamConnection conn = null; conn = (StreamConnection)Connector.open(URL);</pre>
Open a socket connection over direct TCP.	<p>> Invoke <code>Connector.open()</code>, specifying <code>socket</code> as the protocol, appending the <code>deviceside=true</code> parameter to the end of the URL.</p> <pre>private static String URL = "socket:// local_machine_IP:4444;deviceside=true"; StreamConnection conn = null; conn = (StreamConnection)Connector.open(URL);</pre>
Open a socket connection over direct TCP, specifying APN information.	<p>> Invoke <code>Connector.open()</code>, specifying <code>socket</code> as the protocol, appending the <code>deviceside=true</code> parameter to the end of the URL. Specify the following APN parameters:</p> <ul style="list-style-type: none"> • The <code>APN</code> parameter contains the APN over which the connection will be made. • The <code>tunnelauthusername</code> parameter contains the user name to connect to the APN. • The <code>tunnelauthpassword</code> parameter contains the password for the <code>tunnelauthusername</code>. <p>The <code>tunnelauthusername</code> and <code>tunnelauthpassword</code> parameters can be omitted from the connection URL if they are not required by the APN.</p> <p>If you are creating a direct TCP connection, use these parameters. Connections through the BlackBerry® MDS Services are automatically routed by the BlackBerry device; therefore, no APN information is required.</p> <pre>private static String URL = "socket:// local_machine_IP:4444;deviceside=true;apn=internet.com;tunnelauthus ername=user165;tunnelauthpassword=user165password"; StreamConnection conn = null; conn = (StreamConnection)Connector.open(URL);</pre>
Send and receive data.	<p>> Invoke <code>openInputStream()</code> and <code>openOutputStream()</code>.</p> <pre>OutputStreamWriter _out = new OutputStreamWriter(conn.openOutputStream()); String data = "This is a test"; int length = data.length(); _out.write(data, 0, length); InputStreamReader _in = new InputStreamReader(conn.openInputStream()); char[] input = new char[length]; for (int i = 0; i < length; ++i) { input[i] = (char)_in.read(); };</pre>
Close the Socket connection.	<p>> Invoke <code>close()</code> on the input and output streams and the socket connection.</p> <pre>_in.close(); _out.close(); conn.close();</pre> <p>Each of the <code>close()</code> methods throws an <code>IOException</code>. Make sure that the BlackBerry® Java® Application implements exception handling.</p>

Datagram connections

Datagrams are independent packets of data that applications send over networks. A `Datagram` object is a wrapper for the array of bytes that is the payload of the datagram. Use a datagram connection to send and receive datagrams.

Use datagram connections

To use a datagram connection, you must have your own infrastructure to connect to the wireless network, including an APN for GPRS networks. Using UDP connections requires that you work closely with service providers. Verify that your service provider supports UDP connections.

Task	Steps
Before opening a datagram connection, verify that the BlackBerry® device is in network coverage.	<p>> Use the <code>CoverageInfo</code> class and <code>CoverageStatusListener</code> interface of the <code>net.rim.device.api.system</code> package to make sure that the BlackBerry device is in network coverage.</p> <p>Even though the <code>CoverageInfo</code> class and the <code>CoverageStatusListener</code> interface can determine if the BlackBerry device that your BlackBerry Java® Application is on is in network coverage, they cannot guarantee that a subsequent network connection will be successful.</p>
Open a datagram connection.	<p>1. Invoke <code>Connector.open()</code>, specifying udp as the protocol.</p> <p>2. Cast the returned object as a <code>DatagramConnection</code> object.</p> <pre>(DatagramConnection)Connector.open("udp://host:dest_port[:src_port]/apn");</pre> <p>where:</p> <ul style="list-style-type: none"> host is the host address in dotted ASCII-decimal format. dest-port is the destination port at the host address (optional for receiving messages). src-port is the local source port (optional). apn is the network APN in string format.
Receive datagrams from all ports at the specified host.	> Omit the destination port in the connection string.
Open a datagram connection on a non-GPRS network.	<p>> Specify the source port number, including the trailing slash mark.</p> <p>For example, the address for a CDMA network connection would be <code>udp://121.0.0.0:2332;6343/</code>.</p> <p>You can send and receive datagrams on the same port.</p>
Create a datagram.	<p>> Invoke <code>DatagramConnection.newDatagram()</code>.</p> <pre>Datagram outDatagram = conn.newDatagram(buf, buf.length);</pre>
Add data to a diagram.	<p>> Invoke <code>Datagram.setData()</code>.</p> <pre>byte[] buf = new byte[256]; outDatagram.setData(buf, buf.length);</pre>
Send data on the datagram connection.	<p>> Invoke <code>send()</code> on the datagram connection.</p> <pre>conn.send(outDatagram);</pre> <p>If a BlackBerry® Java® Application attempts to send a datagram on a datagram connection and the recipient is not listening on the specified source port, an <code>IOException</code> is thrown. Make sure that the BlackBerry Java Application implements exception handling.</p>

Task	Steps
Receive data on the datagram connection.	<ul style="list-style-type: none"> > Invoke <code>receive()</code> on the datagram connection. Since the <code>receive()</code> method blocks other operations until it receives a data packet, use a timer to retransmit the request or close the connection if a reply does not arrive. <pre>byte[] buf = new byte[256]; Datagram inDatagram = conn.newDatagram(buf, buf.length); conn.receive(inDatagram);</pre>
Extract data from a datagram.	<ul style="list-style-type: none"> > Invoke <code>getData()</code>. If you know the type of data that you are receiving, convert the data to the appropriate format. <pre>String received = new String(inDatagram.getData());</pre>
Close the datagram connection.	<ul style="list-style-type: none"> > Invoke <code>close()</code> on the input and output streams, and on the datagram connection object.

Using port connections

Using a serial or USB connection, BlackBerry® Java® Applications can communicate with desktop applications when they are connected to a computer using a serial or USB port. This type of connection also lets BlackBerry Java Applications communicate with a peripheral device that plugs into the serial or USB port.

Use USB or serial port connections

Task	Steps
Open a USB or serial port connection.	<ul style="list-style-type: none"> > Invoke <code>Connector.open()</code>, specifying comm as the protocol and COM1 or USB as the port. <pre>private StreamConnection _conn = (StreamConnection)Connector.open("comm:COM1;baudrate=9600;bitsperchar=8;parity=none;stopbits=1");</pre>
Send data on the USB or serial port connection.	<ol style="list-style-type: none"> 1. Invoke <code>openDataOutputStream()</code> or <code>openOutputStream()</code>. <pre>DataOutputStream _dout = _conn.openDataOutputStream();</pre> 2. Use the write methods on the output stream to write data. <pre>private String data = "This is a test"; _dout.writeChars(data);</pre>
Receive data on the USB or serial port connection.	<p>Use a non-main event thread to read data from the input stream.</p> <ol style="list-style-type: none"> 1. Invoke <code>openInputStream()</code> or <code>openDataInputStream()</code>. <pre>DataInputStream _din = _conn.openInputStream();</pre> 2. Use the read methods on the input stream to read data. <pre>String contents = _din.readUTF();</pre>
Close the USB or serial port connection.	<ol style="list-style-type: none"> 1. Invoke <code>close()</code> on the input and output streams, and on the port connection object. 2. The <code>close()</code> method can throw <code>IOExceptions</code>. Make sure the BlackBerry® Java® Application implements exception handling. <pre>_din.close(); _dout.close(); conn.close();</pre>

Use Bluetooth serial port connections

You can use the Bluetooth® API (`net.rim.device.api.bluetooth`) to let your BlackBerry® Java® Application access the Bluetooth Serial Port Profile, part of the JSR 82 implementation, and initiate a server or client Bluetooth serial port connection to a computer or other Bluetooth enabled device.

Task	Steps
Open a Bluetooth connection.	<ul style="list-style-type: none"> > Invoke <code>Connector.open()</code>, providing the serial port information that <code>BluetoothSerialPort.getSerialPortInfo()</code> returns as a parameter. <pre>BluetoothSerialPortInfo[] info = BluetoothSerialPort.getSerialPortInfo(); StreamConnection _bluetoothConnection = (StreamConnection)Connector.open(info[0].toString(), Connector.READ_WRITE);</pre>
Send data on the Bluetooth connection.	<ol style="list-style-type: none"> 1. Invoke <code>openDataOutputStream()</code> or <code>openOutputStream()</code>. <pre>DataOutputStream _dout = _bluetoothConnection.openDataOutputStream();</pre> 2. Use the write methods on the output stream to write data. <pre>private static final int JUST_OPEN = 4; _dout.writeInt(JUST_OPEN);</pre>

Task	Steps
Receive data on the Bluetooth connection.	<ol style="list-style-type: none">1. Create a non-main event thread to read data from the input stream.2. Invoke <code>openInputStream()</code> or <code>openDataInputStream()</code>. <code>DataInputStream _din = _bluetoothConnection.openDataInputStream();</code>3. Use the read methods on the input stream to read the data. <code>String contents = _din.readUTF();</code>
Close the Bluetooth connection.	<ol style="list-style-type: none">1. Invoke <code>close()</code> on the input and output streams, and on the Bluetooth serial port connection object.2. The <code>close()</code> method can throw <code>IOExceptions</code>. Make sure the BlackBerry® Java® Application implements exception handling. <pre>if (_bluetoothConnection != null) { try { _bluetoothConnection.close(); } catch(IOException ioe) { } } if (_din != null) { try { _din.close(); } catch(IOException ioe) { } } if (_dout != null) { try { _dout.close(); } catch(IOException ioe) { } } _bluetoothConnection = null; _din = null; _dout = null;</pre>

See “Code sample: Listening for data on the serial port and rendering the data when it arrives” on page 124 for more information.

Code samples

Code sample: Using an HTTP connection to retrieve data

The `HTTPFetch.java` example requires that you create resource files in the BlackBerry® Java® Application project and define the required resource keys. See “Localizing BlackBerry Java Applications” on page 223 for more information on creating resource files.

Example: HTTPFetch.java

```
/**
 * HTTPFetch.java
```

```

* Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
*/

package com.rim.samples.docs.httpfetch;

import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.system.*;
import javax.microedition.io.*;
import java.io.*;
import com.rim.samples.docs.resource.*;

public class HTTPFetch extends UiApplication implements HTTPFetchResource
{
    // Constants.
    private static final String SAMPLE_PAGE = "http://localhost/testpage/sample.txt";
    private static final String[] HTTP_PROTOCOL = {"http://", "http:\\"};

    // Members.
    private MainScreen _mainScreen;
    private RichTextField _content;

    /**
     * Send and receive data over the network on a
     * separate thread from the main thread of your application.
     */
    ConnectionThread _connectionThread = new ConnectionThread();

    //statics
    private static ResourceBundle _resources = ResourceBundle.getBundle(
        HTTPFetchResource.BUNDLE_ID, HTTPFetchResource.BUNDLE_NAME);

    public static void main(String[] args) {
        HTTPFetch theApp = new HTTPFetch();
        theApp.enterEventDispatcher();
    }

    /**
     * The ConnectionThread class manages the HTTP connection.
     * Fetch operations are not queued, but if a second fetch request
     * is made while a previous request is still active,
     * the second request stalls until the previous request completes.
     */
    private class ConnectionThread extends Thread {
        private static final int TIMEOUT = 500; //ms

        private String _theUrl;

        /* The volatile keyword indicates that because the data is shared,
         * the value of each variable must always be read and written from memory,
         * instead of cached by the VM. This technique is equivalent to wrapping
         * the shared data in a synchronized block, but produces less overhead.
         */
        private volatile boolean _start = false;
        private volatile boolean _stop = false;
    }
}

```

```

/**
 * Retrieve the URL. The synchronized keyword makes sure that only one
 * thread at a time can call this method on a ConnectionThread object.
 */
public synchronized String getUrl() {
    return _theUrl;
}

/**
 * Fetch a page. This method is invoked on the connection thread by
 * fetchPage(), which is invoked in the application constructor when
 * the user selects the Fetch menu item.
 */
public void fetch(String url) {
    _start = true;
    _theUrl = url;
}

/**
 * Close the thread. Invoked when the application exits.
 */
public void stop() {
    _stop = true;
}

/**
 * Open an input stream and extract data. Invoked when the thread
 * is started.
 */
public void run() {
    for(;;) {
        // Thread control.
        while( !_start && !_stop) {
            // No connections are open for fetch requests,
            // but the thread has not been stopped.
            try {
                sleep(TIMEOUT);
            } catch (InterruptedException e) {
                System.err.println(e.toString());
            }
        }
        // Exit condition.
        if ( _stop ) {
            return;
        }
        /* Ensure that fetch requests are not missed
         * while received data is processed.
         */
        synchronized(this) {
            // Open the connection and extract the data.
            StreamConnection s = null;
            try {
                s = (StreamConnection)Connector.open(getUrl());
                InputStream input = s.openInputStream();
                // Extract data in 256 byte chunks.
                byte[] data = new byte[256];
                int len = 0;
                StringBuffer raw = new StringBuffer();

```

```

        while ( -1 != (len = input.read(data)) ) {
            raw.append(new String(data, 0, len));
        }
        String text = raw.toString();

        updateContent(text);

        input.close();
        s.close();
    } catch (IOException e) {
        System.err.println(e.toString());
        // Display the text on the screen.
        updateContent(e.toString());
    }
    // Reset the start state.
    _start = false;
}
}
}

private final class HTTPMainScreen extends MainScreen
{
    // Close the connection thread when the user closes the application.
    public void close() {
        _connectionThread.stop();
        super.close();
    }
}

// Constructor.
public HTTPFetch() {
    _mainScreen = new HTTPMainScreen();
    _mainScreen.setTitle(new LabelField(
        _resources.getString(APPLICATION_TITLE), LabelField.ELLIPSIS
        | LabelField.USE_ALL_WIDTH));
    _mainScreen.add(new SeparatorField());
    _content = new RichTextField(
        _resources.getString(HTTPDEMO_CONTENT_DEFAULT));
    _mainScreen.add(_content);

    // Start the helper thread.
    _connectionThread.start();
    pushScreen(_mainScreen);
    fetchPage(SAMPLE_PAGE);
}

// Retrieve web content.
private void fetchPage(String url) {
    // Perform basic validation (set characters to lowercase and add http:// or https://
/).
    String lcase = url.toLowerCase();
    boolean validHeader = false;
    int i = 0;
    for (i = HTTP_PROTOCOL.length - 1; i >= 0; --i) {
        if ( -1 != lcase.indexOf(HTTP_PROTOCOL[i]) ) {
            validHeader = true;
            break;

```

```

    }
    if ( !validHeader ) {
        // Prepend the protocol specifier if it is missing.
        url = HTTP_PROTOCOL[0] + url;
    }

    // Create a new thread for connection operations.
    _connectionThread.fetch(url);
}
// Display the content.
private void updateContent(final String text) {
    /* This technique creates several short-lived objects but avoids
    * the threading issues involved in creating a static Runnable and
    * setting the text.
    */
    UiApplication.getUiApplication().invokeLater(new Runnable() {
        public void run() {
            _content.setText(text);
        }
    });
}
}
}

```

Code sample: Listening for data on the serial port and rendering the data when it arrives

Example: BluetoothSerialPortDemo.java

```

/**
 * BluetoothSerialPortDemo.java
 * Copyright (C) 2004-2005 Research In Motion Limited.
 */

/* The client side of a simple serial port demonstration application.
 * This application listens for text on the serial port and
 * renders the data when it arrives.
 */

package com.rim.samples.docs.bluetoothserialportdemo;

import java.io.*;
import javax.microedition.io.*;
import net.rim.device.api.bluetooth.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;
import com.rim.samples.docs.resource.*;

```

```

public class BluetoothSerialPortDemo extends UiApplication implements
BluetoothSerialPortDemoResResource
{
    //statics -----
    private static ResourceBundle _resources;

    private static final int INSERT = 1;
    private static final int REMOVE = 2;
    private static final int CHANGE = 3;
    private static final int JUST_OPEN = 4;
    private static final int CONTENTS = 5;
    private static final int NO_CONTENTS = 6;

    static {
        _resources = ResourceBundle.getBundle(BluetoothSerialPortDemoResResource.BUNDLE_ID,
BluetoothSerialPortDemoResResource.BUNDLE_NAME);
    }

    private EditField _infoField;
    private StreamConnection _bluetoothConnection;
    private DataInputStream _din;
    private DataOutputStream _dout;

    private final class BluetoothDemoScreen extends MainScreen
    {
        protected void makeMenu(Menu menu, int instance)
        {
            if (_infoField.getTextLength() > 0) {
                menu.add(new MenuItem(_resources, MENUITEM_COPY, 100000, 10) {
                    public void run() {
                        Clipboard.getClipboard().put(_infoField.getText());
                    }
                });
            }
            super.makeMenu(menu, instance);
        }

        public void close()
        {
            closePort();
            super.close();
        }
    }

    public static void main(String[] args)
    {
        new BluetoothSerialPortDemo().enterEventDispatcher();
    }
}

```

```

//constructor -----
public BluetoothSerialPortDemo()
{
    MainScreen mainScreen = new BluetoothDemoScreen();
    mainScreen.setTitle(new LabelField(_resources.getString(TITLE),
LabelField.USE_ALL_WIDTH));

    _infoField = new EditField(Field.READONLY);
    mainScreen.add(_infoField);

    pushScreen(mainScreen);

    invokeLater(new Runnable() {
        public void run() {
            openPort();
        }
    });
}

// Close the serial port.
private void closePort() {
    if (_bluetoothConnection != null) {
        try {
            _bluetoothConnection.close();
        } catch(IOException ioe) {
        }
    }
    if (_din != null) {
        try {
            _din.close();
        } catch(IOException ioe) {
        }
    }
    if (_dout != null) {
        try {
            _dout.close();
        } catch(IOException ioe) {
        }
    }
    _bluetoothConnection = null;
    _din = null;
    _dout = null;
}

// Open the serial port.
private void openPort() {
    if (_bluetoothConnection != null) {
        closePort();
    }
}

```

```

        new InputThread().start();
    }

private class InputThread extends Thread {

public void run() {
try {
BluetoothSerialPortInfo[] info = BluetoothSerialPort.getSerialPortInfo();
if( info == null || info.length == 0 ) {
    invokeAndWait( new Runnable() {
        public void run() {
            Dialog.alert( "No bluetooth serial ports available for connection." );
            closePort();
            System.exit(1);
        }
    });
}
}

    _bluetoothConnection = (StreamConnection)Connector.open( info[0].toString(),
Connector.READ_WRITE );
    _din = _bluetoothConnection.openDataInputStream();
    _dout = _bluetoothConnection.openDataOutputStream();

    } catch(IOException e) {
    invokeAndWait( new Runnable() {
        public void run() {
            Dialog.alert("Unable to open serial port");
            closePort();
            System.exit(1);
        }
    });
    } catch( UnsupportedOperationException e ) {
    invokeAndWait( new Runnable() {
        public void run() {
            Dialog.alert("This handheld or simulator does not support bluetooth.");
            closePort();
            System.exit(1);
        }
    });
    }

    try {
        int type, offset, count;
        String value;
        _dout.writeInt(JUST_OPEN);
        _dout.flush();
        for (;;) {
            type = _din.readInt();
            if (type == INSERT) {
                offset = _din.readInt();
                value = _din.readUTF();
                insert(value, offset);
            } else if (type == REMOVE) {
                offset = _din.readInt();
                count = _din.readInt();
                remove(offset, count);
            }
        }
    }
}

```

```

        } else if (type == JUST_OPEN) {
            // Send contents to desktop.
            value = _infoField.getText();
            if (value == null || value.equals("")) {
                _dout.writeInt(NO_CONTENTS);
                _dout.flush();
            } else {
                _dout.writeInt(CONTENTS);
                _dout.writeUTF(_infoField.getText());
                _dout.flush();
            }
        } else if (type == CONTENTS) {
            String contents = _din.readUTF();
            synchronized(Application.getEventLock()) {
                _infoField.setText(contents);
            }
        } else if (type == NO_CONTENTS) {
        } else {
            throw new RuntimeException();
        }
    }
} catch(IOException ioe) {
    invokeLater(new Runnable() {
        public void run() {
            Dialog.alert("Problems reading from or writing to serial port.");
            closePort();
            System.exit(1);
        }
    });
}

}

}

private void insert(final String msg, final int offset) {
    invokeLater(new Runnable() {
        public void run() {
            _infoField.setCursorPosition(offset);
            _infoField.insert(msg);
        }
    });
}

private void remove(final int offset, final int count) {
    invokeLater(new Runnable() {
        public void run() {
            _infoField.setCursorPosition(offset+count);
            _infoField.backspace(count);
        }
    });
}

}

```


Working with Wi-Fi connections on a BlackBerry device

Work with wireless access families
Work with a Wi-Fi connection

Work with wireless access families

Working with the BlackBerry® device transceiver involves using APIs that make reference to wireless access families.

See the API reference for the BlackBerry Java Development Environment for more information about wireless access families.

Wireless access family	Description
3GPP	includes GPRS, EDGE, UMTS, GERAN, UTRAN, and GAN
CDMA	includes CDMA1x and EVDO
WLAN	includes 802.11™, 802.11a™, 802.11b™, 802.11g™

Identify the wireless access families that a BlackBerry device supports

Task	Steps
Retrieve the wireless access families that a BlackBerry device supports.	> Invoke <code>RadioInfo.getSupportedWAFs()</code> .
Determine if a BlackBerry device supports one or more wireless access families.	> Invoke <code>RadioInfo.areWAFsSupported(int wafs)</code> .
Determine the wireless access family transceivers that are turned on.	> Invoke <code>RadioInfo.getActiveWAFs()</code> .

Turn on a transceiver for a wireless access family

Task	Steps
Turn on the transceiver for a wireless access family.	> Invoke <code>Radio.activateWAFs(int WAFs)</code> . The WAFs parameter is a bitmask.
Turn off the transceiver for a wireless access family.	> Invoke <code>Radio.deactivateWAFs(int WAFs)</code> . The WAFs parameter is a bitmask.

Receive notifications of transceiver events

Task	Steps
Enable a BlackBerry Application to receive transceiver events from multiple transceivers.	> Register a transceiver event listener for specific wireless access families by invoking <code>Application.addRadioListener(int wafFilter, RadioListener listener)</code> . This method registers the listener to listen for events from the wireless access families specified in the <code>wafFilter</code> parameter. This parameter is applied only to <code>RadioStatusListeners</code> .
Determine the wireless access family that generated a transceiver event.	<p>If a BlackBerry Application registers a <code>RadioStatusListener</code> method for more than one wireless access family, when the <code>RadioStatusListener</code> method notifies the application of a transceiver event, the application will not be able to determine the wireless access family that generated the transceiver event.</p> <p>> Register a unique <code>RadioStatusListener</code> instance for each wireless access family.</p>
Receive notifications of Wi-Fi transceiver events	> Invoke <code>Application.addRadioListener(int wafFilter, RadioListener listener)</code> using the <code>RadioInfo.WAF_WLAN</code> parameter.
Receive notifications when the transceiver for the WLAN wireless access family connects or disconnects with a wireless access point.	<ol style="list-style-type: none"> 1. From the <code>net.rim.device.api.system</code> package, import the <code>WLANListener</code> and <code>WLANConnectionListener</code> interfaces and the <code>WLANInfo</code> class. 2. Invoke <code>Application.addRadioListener(int wafFilter, RadioListener listener)</code> using the <code>RadioInfo.WAF_WLAN</code> field and a <code>RadioStatusListener</code> object as parameters. <pre>RadioStatusListener listener = new RadioStatusListener() { ... }; Application.addRadioListener(RadioInfo.WAF_WLAN, listener);</pre> 3. Register a <code>WLANConnectionListener</code> object by invoking the <code>WLANInfo.addListener()</code> method using a <code>WLANConnectionListener</code> object as a parameter. <pre>WLANConnectionListener listener = new WLANConnectionListener() {...}; WLANInfo.addListener(listener);</pre>

Work with a Wi-Fi connection

Query the status of the Wi-Fi transceiver on a BlackBerry device

Task	Steps
Determine if the transceiver for the WLAN wireless access family is on.	> Create an IF statement that tests the value of <code>RadioInfo.WAF_WLAN</code> and the value returned by <code>RadioInfo.getActiveWAFs()</code> , for example: <pre>if ((RadioInfo.getActiveWAFs() & RadioInfo.WAF_WLAN) != 0) { ... }</pre>

Task	Steps
Determine if the transceiver for the WLAN wireless access family is connected to a wireless access point.	<ol style="list-style-type: none"> 1. From the <code>net.rim.device.api.system</code> package, import the <code>WLANInfo</code> class. 2. Create an IF statement that tests the value of <code>WLANInfo.WLAN_STATE_CONNECTED</code> and the value returned by <code>WLANInfo.getWLANState()</code>. <pre>if (WLANInfo.getWLANState() == WLANInfo.WLAN_STATE_CONNECTED) {...}</pre> <p>The <code>WLANInfo.getWLANState()</code> method checks if a BlackBerry device has an IP address and can transfer data over a Wi-Fi network. If the transceiver for the WLAN wireless access family is off, this method returns <code>WLANInfo.WLAN_STATE_DISCONNECTED</code>.</p>
Retrieve status information about a wireless access point or the active Wi-Fi profile.	<p>You can let a BlackBerry device application retrieve status information such as the data rate of the connection, the wireless LAN standards used (802.11a, b or g), the SSID of the associated access point, or the name of the Wi-Fi profile in use.</p> <p>The transceiver for the WLAN wireless access family must be connected to a wireless access point.</p> <ol style="list-style-type: none"> 1. From the <code>net.rim.device.api.system</code> package, import the <code>WLANInfo</code> class. 2. Invoke <code>WLANInfo.getAPIInfo()</code>, storing a reference to <code>WLANInfo.WLANAPIInfo</code> that this method returns. The <code>WLANInfo.WLANAPIInfo</code> object contains a snapshot of the current wireless network. <pre>WLANInfo.WLANAPIInfo info = WLANInfo.getAPIInfo();</pre> <p>If the BlackBerry device is not connected to an access point, the <code>WLANInfo.getAPIInfo()</code> method returns null.</p> <p>See the API reference for the BlackBerry Java Development Environment for more information about <code>WLANInfo.WLANAPIInfo</code>.</p>

Accessing a wireless network through a wireless access point

Task	Steps
Determine if the BlackBerry device is accessing a wireless network through a wireless access point.	<ol style="list-style-type: none"> 1. Invoke the <code>RadioInfo.getNetworkService</code> method using the <code>RadioInfo.WAF_3GPP</code> parameter. 2. In the bitmask of the <code>RadioInfo.NETWORK_SERVICE_*</code> flags that the <code>getNetworkService(int)</code> method returns, check to see if the <code>RadioInfo.NETWORK_SERVICE_GAN</code> flag is set in the return value.
When a 3GPP wireless access family generates a transceiver event, determine if the BlackBerry device is accessing a wireless network through a wireless access point.	<p>> When the listener's <code>RadioStatusListener.networkServiceChange(int networked, int service)</code> method is invoked, check for the <code>RadioInfo.NETWORK_SERVICE_GAN</code> flag in the service parameter. If this flag is set in the service parameter, the BlackBerry device is accessing a wireless network through a wireless access point.</p>

Determine if the BlackBerry device is in a wireless coverage area

Task	Steps
Receive notifications of changes in the connectivity state of a BlackBerry device.	> Use the <code>addListener()</code> methods of the <code>CoverageInfo</code> class.
Determine if the BlackBerry device has enough wireless coverage to attempt a direct TCP connection through a wireless access point.	> Invoke <code>isCoverageSufficient(COVERAGE_CARRIER, RadioInfo.WAF_WLAN, false)</code> .
Determine if the BlackBerry device has enough wireless coverage to attempt a WLAN enterprise connection through a wireless access point.	> Invoke <code>isCoverageSufficient(COVERAGE_MDS, RadioInfo.WAF_WLAN, false)</code> .

Open a Wi-Fi connection

The `interface=wifi` parameter applies only to TCP/UDP connections. To establish a Wi-Fi® connection and use a Wi-Fi API in a BlackBerry device application, the wireless service provider must support Wi-Fi access.

Task	Steps
Open a Wi-Fi socket connection.	> Invoke <code>Connector.open()</code> , specify socket as the protocol, and append the <code>deviceside=true</code> parameter and the <code>interface=wifi</code> parameter to the end of the URL string value. <pre>private static String URL = "socket:// local_machine_IP:4444;deviceside=true;interface=wifi"; StreamConnection conn = null; conn = (StreamConnection)Connector.open(URL);</pre>
Open a Wi-Fi HTTP connection.	1. Invoke <code>Connector.open()</code> , specify http as the protocol, and append the <code>interface=wifi</code> parameter to the end of the URL string value. 2. Cast the returned object as an <code>HttpConnection</code> or a <code>StreamConnection</code> object. <pre>HttpConnection conn = null; String URL = "http://www.myServer.com/ myContent;deviceside=true;interface=wifi"; conn = (HttpConnection)Connector.open(URL);</pre>
Open a Wi-Fi HTTPS connection.	1. Invoke <code>Connector.open()</code> , specify https as the protocol, and append the <code>interface=wifi</code> parameter to the end of the URL string value. 2. Cast the returned object as an <code>HttpsConnection</code> object. <pre>HttpsConnection conn = null; String URL = "https://host:443/;deviceside=true;interface=wifi"; conn = (HttpsConnection)Connector.open(URL);</pre>

Creating notifications

- Types of notification events
- Add a new event source
- Respond to deferred events
- Cancel events
- Customize system notifications for immediate events
- Code samples

Types of notification events

The notification API (`net.rim.device.api.notification`) lets you add custom events for your BlackBerry® Java® Application and define the type of notification that BlackBerry device users receive when custom events occur.

Notification event type	Description
Immediate events	With immediate events, BlackBerry® devices notify the BlackBerry device user as soon as the event occurs, using a system notification, such as a flashing LED, vibration, or tune. A BlackBerry Java® Application cannot request a specific type of notification. In the BlackBerry device profiles list, BlackBerry device users control how they receive notification of immediate events by choosing an active profile and setting profile options.
Deferred events	With deferred events, BlackBerry® devices schedule events in a queue according to their priority. When the event occurs, BlackBerry Java® Applications that are affected by the event can provide custom notifications to the BlackBerry device user, typically by displaying a UI element, such as a dialog box. BlackBerry devices do not provide system-wide notifications for deferred events.

Add a new event source

Task	Steps
Create a unique long ID.	<ol style="list-style-type: none"> 1. Define a long ID for each notification event. <pre>public static final long ID_1 = 0xdc5bf2f81374095L;</pre> 2. Open the BlackBerry® Integrated Development Environment. 3. In the BlackBerry IDE text pane, type a string. 4. Select the string. 5. Right-click the highlighted string. 6. Click Convert "string" to Long.
Define a source object.	<ol style="list-style-type: none"> 1. Define an object that provides the source for the event. 2. Your implementation of <code>toString()</code> returns the string to display in the profiles list. <pre>Object event = new Object() { public String toString() { return "Notification Demo"; } }</pre>
Add your BlackBerry® Java® Application to the BlackBerry device profiles.	<ol style="list-style-type: none"> 1. Invoke <code>NotificationsManager.registerSource()</code>. 2. In this method, specify a unique event ID, the source object, and, for deferred events only, one of the following priority levels: <ul style="list-style-type: none"> • <code>NotificationsConstants.CRITICAL</code> • <code>NotificationsConstants.SENSITIVE</code> • <code>NotificationsConstants.IMPORTANT</code> • <code>NotificationsConstants.DEFAULT_LEVEL</code> • <code>NotificationsConstants.CASUAL</code>

Register the event source when the BlackBerry device starts

To register the event source when the BlackBerry® device starts, create a separate project that acts as an alternative entry point to the main BlackBerry Java® Application. When the BlackBerry device starts, this project automatically runs as a system module and passes an argument to the BlackBerry Java Application, allowing the BlackBerry Java Application to perform any one-time initializations.

Task	Steps
Create an initialization project.	<ol style="list-style-type: none"> 1. In the BlackBerry® Integrated Development Environment, create a project. 2. Right-click the project, and then click Properties. 3. On the Application tab, in the Project type drop-down list, click Alternate CLDC Application Entry Point. 4. In the Alternate entry point drop-down list, click the event source project. 5. In the Arguments passed to field, type autostartup. 6. Select the Auto-run on startup option. 7. Select the System module option. 8. Click OK.

Task	Steps
Perform initializations at the alternative entry point.	<p>Make sure that the string checked in the If statement matches the value you type in the Arguments passed to field in the BlackBerry® IDE project.</p> <pre>> In your main() method, perform any required initializations. public static void main (String[] args) { if (args.length > 0 && args[0].equals("autostartup")) { //BlackBerry Java® Application runs as a system module at startup. //Perform any necessary one-time automatic initialization. } else { //BlackBerry Java Application is being run by a user. } }</pre>

See "Code sample: Add a new event source" on page 141 for more information.

Triggering events

Task	Steps
Trigger an immediate event.	<pre>> Invoke triggerImmediateEvent(). NotificationsManager.triggerImmediateEvent(ID_1, 0, this, null);</pre>
Trigger a deferred event.	<ol style="list-style-type: none"> Invoke negotiateDeferredEvent(). <pre>NotificationsManager.negotiateDeferredEvent(ID_1, 0, this, -1, NotificationsConstants.MANUAL_TRIGGER, null);</pre> If you invoke negotiateDeferredEvent(long, long, Object, long, int, Object), your BlackBerry® Java® Application must implement the NotificationEngineListener to receive events and respond appropriately.

Respond to deferred events

Task	Steps
Provide a custom UI notification.	<p>> Implement the <code>NotificationsEngineListener</code> interface.</p> <pre>private static class ListenerImpl implements NotificationsEngineListener {...}</pre>
Define behavior if an event is superseded by another event at the same or higher priority.	<p>> Implement <code>deferredEventWasSuperseded()</code>.</p> <pre>public void deferredEventWasSuperseded(long sourceID, long eventID, Object eventReference, Object context) { final long _eventID = eventID; er = eventReference; _app.invokeLater(new Runnable() { public void run() { NotificationsManager.cancelDeferredEvent(ID_1, _eventID, er, NotificationsConstants.MANUAL_TRIGGER, null); } }); }</pre>
Define behavior if the BlackBerry® device user inserts or removes the BlackBerry device from the holster.	<p>> Implement <code>notificationsEngineStateChanged()</code>.</p> <pre>public void notificationsEngineStateChanged(int stateInt, long sourceID, long eventID, Object eventReference, Object context) { if(stateInt == NotificationsConstants.OUT_OF_HOLSTER_ENGINE_STATE) { // Perform action if the BlackBerry device is removed from the holster. } if(stateInt == NotificationsConstants.IN_HOLSTER_ENGINE_STATE) { // Perform action if the BlackBerry device is inserted into the holster. } }</pre>
Define notification when the event occurs.	<p>> Implement <code>proceedWithDeferredEvent()</code>.</p> <pre>public void proceedWithDeferredEvent(long sourceID, long eventID, Object eventReference, Object context) { final long _eventID = eventID; _app.invokeLater(new Runnable() { public void run() { String s = "This event has occurred: " + _eventID; Dialog d = new Dialog(Dialog.D_OK, s, Dialog.OK, Bitmap.getPredefinedBitmap(Bitmap.INFORMATION), 0); d.show(); _eventHashtable.put(_eventID, d); } }); }</pre>
Register the notifications listener with the <code>NotificationsManager</code> .	<p>You can register only one <code>NotificationsEngineListener</code> for each BlackBerry® Java® Application.</p> <p>> Invoke <code>NotificationsManager.registerNotificationsEngineListener(int, NotificationsEngineListener)</code>, providing as parameters the event source ID of your BlackBerry Java Application and an instance of the class that implements the <code>NotificationsEngineListener</code> interface.</p> <pre>NotificationsManager.registerNotificationsEngineListener(ID_1, new ListenerImpl(this));</pre>

Cancel events

Task	Steps
Cancel an immediate event.	<ul style="list-style-type: none"> > Invoke <code>cancelImmediateEvent(long, long, Object, Object)</code>, and then the source and event ID. <pre>NotificationsManager.cancelImmediateEvent(ID_1, 0, this, null);</pre>
Cancel a deferred event.	<ul style="list-style-type: none"> > Invoke <code>cancelDeferredEvent(long, long, Object, int, Object)</code>, and then the source and event ID. <pre>NotificationsManager.cancelDeferredEvent(ID_1, 0, this, NotificationsConstants.MANUAL_TRIGGER, null);</pre>
Cancel all deferred events.	<p>If you invoke <code>negotiateDeferredEvent()</code> and do not specify a timeout, you must invoke <code>cancelDeferredEvent()</code> to cancel the event, or the event never expires.</p> <ul style="list-style-type: none"> > Invoke <code>cancelAllDeferredEvents(long, int, Object)</code> to cancel all deferred events that your BlackBerry® Java® Application starts. <pre>NotificationsManager.cancelAllDeferredEvents(ID_1, NotificationsConstants.MANUAL_TRIGGER, null);</pre>

Customize system notifications for immediate events

Task	Steps
Respond to notification events.	<ul style="list-style-type: none"> > Create a class that implements the <code>Consequence</code> and <code>SyncConverter</code> interfaces. The <code>SyncConverter</code> interface defines the functionality necessary to convert data from object to serialized format. <pre>private static class ConsequenceImpl implements Consequence, SyncConverter {...}</pre>
Define a unique ID.	<ul style="list-style-type: none"> > Define a unique ID for the consequence <pre>public static final long ID = 0xbd2350c0dfda2a51L;</pre>
Define the constants.	<ul style="list-style-type: none"> > Declare the DATA and TYPE constants to identify data for the BlackBerry® Java® Application. When the BlackBerry Java Application invokes <code>convert()</code>, the constants identify the type of incoming data from the <code>SyncConverter</code>. <pre>private static final int TYPE = 'n' << 24 'o' << 16 't' << 8 'd'; private static final byte[] DATA = new byte[] { 'm', 'y', '-', 'c', 'o', 'n', 'f', 'i', 'g', '-', 'o', 'b', 'j', 'e', 'c', 't' }; private static final Configuration CONFIG = new Configuration(DATA);</pre>
Create a tune that plays when the BlackBerry® device user receives the notification.	<ul style="list-style-type: none"> > Create a tune that plays as part of the consequence for event notifications. <pre>private static final short BFlat = 466; // 466.16 private static final short TEMPO = 125; private static final short d16 = 1 * TEMPO; private static final short dpause = 10; // 10 millisecond pause private static final short[] TUNE = new short[] { BFlat, d16, dpause, BFlat }; private static final int VOLUME = 80; // Percentage volume.</pre>

Task	Steps
Define a notification.	<pre>> Implement startNotification(). public void startNotification(long consequenceID, long sourceID, long eventID, Object configuration, Object context) { LED.setConfiguration(500, 250, LED.BRIGHTNESS_50); LED.setState(LED.STATE_BLINKING); Alert.startAudio(TUNE, VOLUME); Alert.startBuzzer(TUNE, VOLUME); }</pre>
Stop a notification.	<pre>> Implement stopNotification(). public void stopNotification(long consequenceID, long sourceID, long eventID, Object configuration, Object context) { LED.setState(LED.STATE_OFF); Alert.stopAudio(); Alert.stopBuzzer(); }</pre>
Store the event notification user profile settings.	<pre>> Implement newConfiguration(). public Object newConfiguration(long consequenceID, long sourceID, byte profileIndex, int level, Object context) { return CONFIG; }</pre>
Enable data backup for the event notification user profile settings.	<pre>> Implement SyncConverter.convert(). public SyncObject convert(DataBuffer data, int version, int UID) { try { int type = data.readInt(); int length = data.readCompressedInt(); if (type == TYPE) { byte[] rawdata = new byte[length]; data.readFully(rawdata); return new Configuration(rawdata); } } catch (EOFException e) { System.err.println(e); } return null; }</pre>
Enable data restore for the event notification user profile settings.	<pre>> Implement SyncConverter.convert(). public boolean convert(SyncObject object, DataBuffer buffer, int version) { boolean retval = false; if (object instanceof Configuration) { Configuration c = (Configuration)object; buffer.writeInt(TYPE); buffer.writeCompressedInt(c._data.length); buffer.write(c._data); retval = true; } return retval; }</pre>

Task	Steps
Define the notification configuration.	<ol style="list-style-type: none"> 1. Create a class that implements <code>SyncObject</code> and <code>Persistable</code>. 2. In the class, make sure the <code>SyncObject.getUID()</code> method returns 0 if data synchronization is not required. <pre>private static final class Configuration implements SyncObject, Persistable { public byte[] _data; public Configuration(byte[] data) { _data = data; } public int getUID() { return 0; } }</pre>
Register a custom notification in the <code>NotificationsManager</code> .	<p>> If you create a custom <code>Consequence</code> implementation, register it with the <code>NotificationsManager</code> by invoking <code>registerNotificationsObjects(long, Consequence)</code>.</p> <pre>NotificationsManager.registerConsequence(ConsequenceImpl.ID, new ConsequenceImpl());</pre>

See “Code sample: Creating a custom notification” on page 144 for more information.

Code samples

Code sample: Add a new event source

Example: `NotificationsDemo.java`

```
/**
 * NotificationsDemo.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.notifications;

import net.rim.device.api.notification.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;

public class NotificationsDemo extends UiApplication
{
    public static final long ID_1 = 0xdc5bf2f81374095L;
    private long _eventIdGenerator;
    private static Object er;

    public static final Object event = new Object() {
        public String toString() {
```

```

        return "Sample Notification Event #1";
    }
};

public static void main(String[] args) {
    if ( args.length > 0 && args[0].equals( "autostartup" ) ) {
        NotificationsManager.registerSource(ID_1, event,
NotificationsConstants.CASUAL);
        NotificationsManager.registerConsequence(ConsequenceDemo.ID, new
ConsequenceDemo());
    } else {
        NotificationsDemo app = new NotificationsDemo();
        app.enterEventDispatcher();
    }
}

public NotificationsDemo() {
    MainScreen mainScreen = new NotificationsMainScreen();
    mainScreen.setTitle("Notification Demo App");
    NotificationsManager.registerNotificationsEngineListener(ID_1,
        new NotificationsEngineListenerImpl(this));
    pushScreen(mainScreen);
}

private MenuItem triggerItem = new MenuItem(null, 0, 100, 10) {
    public void run() {
        NotificationsManager.triggerImmediateEvent(ID_1, 0, this, null);
    }
    public String toString() {
        return "Trigger event";
    }
};

private MenuItem deferItem = new MenuItem(null, 0, 100, 10) {
    public void run() {
        long timeout = -1; // Ignored unless trigger is OUT_OF_HOLSTER_TRIGGER.
        int trigger = NotificationsConstants.MANUAL_TRIGGER;
        Object er = new Object();
        NotificationsManager.negotiateDeferredEvent(ID_1, ++_eventIdGenerator,
            er, timeout, trigger, null);
    }
    public String toString() {
        return "Start deferred event";
    }
};

private MenuItem cancelItem = new MenuItem(null, 0, 100, 10) {
    public void run() {
        int trigger = NotificationsConstants.MANUAL_TRIGGER;
        NotificationsManager.cancelDeferredEvent(ID_1, _eventIdGenerator, er,
            trigger, null);
    }
    public String toString() {
        return "Cancel deferred event";
    }
};

private final class NotificationsMainScreen extends MainScreen

```

```

{
    protected void makeMenu( Menu menu, int instance ) {
        menu.add(triggerItem);
        menu.add(deferItem);
        menu.add(cancelItem);
        super.makeMenu(menu, instance);
    }
}

private static class NotificationsEngineListenerImpl implements
    NotificationsEngineListener {
    private UiApplication _app;
    public NotificationsEngineListenerImpl(UiApplication app) {
        _app = app;
    }

    public void deferredEventWasSuperseded(long sourceID, long eventID,
        Object eventReference, Object context) {
        final long _eventID = eventID;
        er = eventReference;
        _app.invokeLater(new Runnable() {
            public void run() {
                NotificationsManager.cancelDeferredEvent(ID_1, _eventID, er,
                    NotificationsConstants.MANUAL_TRIGGER, null);
            }
        });
    }

    public void notificationsEngineStateChanged(int stateInt, long sourceID,
        long eventID, Object eventReference, Object context) {
        if(stateInt == NotificationsConstants.OUT_OF_HOLSTER_ENGINE_STATE) {
            // Perform some action if handheld is removed from holster.
        }
        if(stateInt == NotificationsConstants.IN_HOLSTER_ENGINE_STATE) {
            // Perform some action if handheld is inserted into holster.
        }
    }

    public void proceedWithDeferredEvent(long sourceID, long eventID,
        Object eventReference, Object context) {
        final long _eventID = eventID;
        _app.invokeLater(new Runnable() {
            public void run() {
                String s = "This event has occurred: " + _eventID;
                Dialog d = new Dialog(Dialog.D_OK, s, Dialog.OK,
                    Bitmap.getPredefinedBitmap(Bitmap.INFORMATION), 0);
                d.show();
            }
        });
    }
}
}

```

Code sample: Creating a custom notification

Example: ConsequenceDemo.java

```

/**
 * ConsequenceDemo.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.notifications;

import net.rim.device.api.synchronization.*;
import net.rim.device.api.notification.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;
import java.io.*;

public class ConsequenceDemo implements Consequence, SyncConverter {

    public static final long ID = 0xbd2350c0dfda2a51L;
    private static final int TYPE = 'n' << 24 | 'o' << 16 | 't' << 8 | 'd';
    private static final byte[] DATA = new byte[] {
        'm', 'y', '-', 'c', 'o', 'n', 'f', 'i',
        'g', '-', 'o', 'b', 'j', 'e', 'c', 't' };

    private static final Configuration CONFIG = new Configuration(DATA);

    private static final short BFlat = 466; // The actual value is 466.16.
    private static final short TEMPO = 125;
    private static final short d16 = 1 * TEMPO;
    private static final short pause = 10; // 10 millisecond pause.
    private static final short[] TUNE = new short[] {BFlat, d16, pause, BFlat};
    private static final int VOLUME = 80; // Percentage volume.

    public void startNotification(long consequenceID, long sourceID, long eventID,
        Object configuration, Object context) {
        LED.setConfiguration(500, 250, LED.BRIGHTNESS_50);
        LED.setState(LED.STATE_BLINKING);

        Alert.startAudio(TUNE, VOLUME);
        Alert.startBuzzer(TUNE, VOLUME);
    }

    public void stopNotification(long consequenceID, long sourceID, long eventID,
        Object configuration, Object context) {
        LED.setState(LED.STATE_OFF);
        Alert.stopAudio();
        Alert.stopBuzzer();
    }

    public Object newConfiguration(long consequenceID, long sourceID,
        byte profileIndex, int level, Object context) {
        return CONFIG;
    }

    public SyncObject convert(DataBuffer data, int version, int UID) {
        try {

```

```

        int type = data.readInt();
        int length = data.readCompressedInt();
        if ( type == TYPE ) {
            byte[] rawdata = new byte[length];
            data.readFully(rawdata);
            return new Configuration(rawdata);
        }
    } catch (EOFException e) {
        System.err.println(e);
    }
    return null;
}

public boolean convert(SyncObject object, DataBuffer buffer, int version) {
    boolean retval = false;
    if ( object instanceof Configuration ) {
        Configuration c = (Configuration)object;
        buffer.writeInt(TYPE);
        buffer.writeCompressedInt(c._data.length);
        buffer.write(c._data);
        retval = true;
    }
    return retval;
}

/* Inner class to store configuration profile. */
private static final class Configuration implements SyncObject, Persistable {

    public byte[] _data;

    public Configuration(byte[] data) {
        _data = data;
    }
    public int getUID() {
        return 0;
    }
}
}

```

Integrating with BlackBerry applications

Invoke BlackBerry applications

Invoke BlackBerry applications

You can create BlackBerry® Java® Applications that invoke BlackBerry device applications such as the email, calendar, phone, maps, browser, and camera applications to perform an action or display information. When the third-party BlackBerry Java Application invokes the BlackBerry device application, the third-party BlackBerry Java Application can make the BlackBerry device application perform an action or display information.

Task	Steps
Start the message application and create a new blank message.	<ul style="list-style-type: none"> > Invoke the <code>invokeApplication()</code> method with the following parameters: <ul style="list-style-type: none"> • <code>Invoke.APP_TYPE_MESSAGES</code>: a constant parameter • <code>MessageArguments</code>: a new <code>MessageArguments</code> object that uses the <code>ARG_NEW_SMS</code> parameter <pre>Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments(MessageArguments.ARG_NEW_SMS));</pre>
Start the calendar.	<ul style="list-style-type: none"> > Invoke <code>Invoke.invokeApplication(APP_TYPE_CALENDAR, CalendarArguments)</code>
Start the phone application.	<ul style="list-style-type: none"> > Invoke <code>Invoke.invokeApplication(APP_TYPE_PHONE, PhoneArguments)</code>.
Start the maps application and display the default map view.	<ul style="list-style-type: none"> > Invoke <code>invokeApplication()</code> using a new <code>MapsArguments</code> object. <pre>Invoke.invokeApplication(Invoke.APP_TYPE_MAPS, new MapsArguments());</pre>

See the *API Reference* for more information about using the `net.rim.blackberry.api.invoke.Invoke` class to invoke BlackBerry device applications.

Related topics

- See "Start the media player from the BlackBerry Browser" on page 48 for more information.
- See "Start the media player with no content" on page 49 for more information.
- See "Start the media player with content" on page 49 for more information.
- See "Create new messages" on page 157 for more information.
- See "Open the address book from your BlackBerry Java Application" on page 172 for more information.
- See "Start the task application from your BlackBerry Java Application" on page 178 for more information.
- See "Use the BlackBerry Maps application" on page 224 for more information.

Managing applications

Application manager

Retrieve information about BlackBerry Java Applications

Register BlackBerry Java Applications when the BlackBerry device starts

Communicate with other BlackBerry Java Applications

Determine the services that are available to BlackBerry Java Applications

Listen for changes to IT policies

Application control

Managing code modules

The runtime store

Share runtime objects

Code sample

Application manager

The BlackBerry® JVM on BlackBerry devices includes an application manager that functions as the central dispatcher of operating system events for other BlackBerry Java® Applications.

The `net.rim.device.api.system.ApplicationManager` class lets BlackBerry Java Applications interact with the application manager to perform the following actions:

- interact with processes, such as retrieving the IDs for foreground BlackBerry Java Applications
- post global events to the system
- run a BlackBerry Java Application immediately or at a specific time

Retrieve information about BlackBerry Java Applications

Task	Steps
Retrieve information about the processes that are running.	<pre>> Invoke ApplicationManager.getVisibleApplications(). ApplicationManager manager = ApplicationManager.getApplicationManager(); ApplicationDescriptor descriptors[] = manager.getVisibleApplications();</pre>
Retrieve descriptions of the objects for the BlackBerry Java Applications that are running.	<pre>> Invoke ApplicationDescriptor.getName(). String appname1 = descriptors[0].getName();</pre>
Retrieve a description of the current BlackBerry Java Application.	<pre>> Invoke ApplicationDescriptor.currentApplicationDescriptor(). ApplicationDescriptor descriptor = ApplicationDescriptor.currentApplicationDescriptor();</pre>

Register BlackBerry Java Applications when the BlackBerry device starts

To register the event source when the BlackBerry® device starts, create a separate project that acts as an alternative entry point to the main BlackBerry Java® Application. When the BlackBerry device starts, this project automatically runs as a system module and passes an argument to the BlackBerry Java Application, allowing the BlackBerry Java Application to perform any one-time initialization. You cannot pass arguments to MIDlet BlackBerry Java Applications when the BlackBerry device starts.

Task	Steps
Retrieve information about the processes that are running.	<ol style="list-style-type: none"> 1. In the BlackBerry® Integrated Development Environment, create a project. 2. Right-click the project, and then click Properties. 3. Click the Application tab. 4. In the Project type drop-down list, click Alternate CLDC Application Entry Point. 5. In the Alternate entry point drop-down list, click the alternate entry point project. 6. In the Arguments passed to field, type autostartup. 7. Select the Auto-run on startup option. 8. Select the System module option. 9. Click OK.
Perform initializations at the alternative entry point.	<p>> In your <code>main()</code> method, perform the required initialization. For example:</p> <pre> public static void main (String[] args) {if (args.length > 0 && args[0].equals("autostartup")) { //Application runs as a system module at startup. //Perform any necessary one-time automatic initialization. } else { //Application is being run by a user.} } </pre>

Communicate with other BlackBerry Java Applications

- > To post a system-level event to other BlackBerry® Java® Applications, invoke one of the `ApplicationManager.postGlobalEvent()` methods.

Determine the services that are available to BlackBerry Java Applications

The service book consists of service records, each of which defines a service on a BlackBerry® device. Service records define the communication protocol (WAP or IPPP), the network gateway, and the configuration information such as browser settings.

- > To let your BlackBerry Java® Application interact with the BlackBerry Infrastructure, use the service book API (`net.rim.device.api.servicebook`).

Listen for changes to IT policies

Task	Steps
Enable a BlackBerry Java Application to use IT policies.	> Implement the <code>GlobalEventListener</code> interface.
Identify changes in IT policies.	> Implement <code>GlobalEventListener.eventOccurred()</code> .

See “Code sample: Listening for changes to IT policies” on page 154 for more information.

Application control

The BlackBerry® Application Control IT policy rules provide administrators with the ability to establish the capabilities of a BlackBerry Java® Application when it runs on a specific BlackBerry device. For example, system administrators can use the BlackBerry Application Control IT policy to make sure that a game that exists on the BlackBerry device cannot access the phone API.



Note: The BlackBerry® Application Control IT policy works only when the BlackBerry device and a BlackBerry Enterprise Server are connected. This IT policy does not apply to BlackBerry devices that use only the BlackBerry Internet Service.

Allow a BlackBerry Java Application to request access to resources

1. Create an instance of the `ApplicationPermissions` class.

```
ApplicationPermissions permissions = new ApplicationPermissions();
```

2. Set the build request to ask for event injection privileges.

```
permissions.addPermission( ApplicationPermissions.PERMISSION_EVENT_INJECTOR );
```

3. Determine the access control settings that the BlackBerry® device user set.

```
if( ApplicationPermissionsManager.getInstance().invokePermissionsRequest( permissions )
) {
    System.out.println( "The user saved equal, or more permissive settings" );
} else {
    System.out.println( "The user saved more restrictive settings" );
}
```

Managing code modules

To retrieve information about and manage code modules on the BlackBerry device, use the `CodeModuleManager` class in the `net.rim.device.api.system` package.

Retrieve module information

Task	Steps
Retrieve a handle for a module.	<p>> Invoke <code>getModuleHandle()</code>, and provide the name of the code module as a parameter.</p> <pre>int handle = CodeModuleManager.getModuleHandle("test_module");</pre>
Retrieve specific information about a module.	<p>> Invoke the methods of the <code>CodeModuleManager</code> class, and provide the module handle as a parameter to these methods.</p> <pre>String name = CodeModuleManager.getModuleName(handle); String vendor = CodeModuleManager.getModuleVendor(handle); String description = CodeModuleManager.getModuleDescription(handle); int version = CodeModuleManager.getModuleVersion(handle); int size = CodeModuleManager.getModuleCodeSize(handle); int timestamp = CodeModuleManager.getModuleTimestamp(handle);</pre>

Retrieve an array of handles for existing modules on a BlackBerry device

```
> Invoke getModuleHandles().
int handles[] = CodeModuleManager.getModuleHandles();
String name = CodeModuleManager.getModuleName( handles[0]);
```

Create code modules

Task	Steps
Create a module without data.	<p>> Invoke <code>createNewModule()</code> and provide the size of the module in bytes as a parameter.</p> <pre>int handle = CodeModuleManager.createNewModule(3000);</pre>
Create a module with data.	<p>> Invoke <code>createNewModule(int, byte[], int)</code>, providing the following parameters:</p> <ul style="list-style-type: none"> the length in bytes of the entire module the byte array to add to the module the length parameter to specify the number of bytes from the byte array to add to the start of the module. <pre>static int createNewModule(int, byte[], int);</pre>
Write data into a module.	<p>You can write data into a code module in increments, as long as you know the offset at which to add data.</p> <p>> Invoke <code>writeNewModule()</code> and provide a byte array of data as a parameter to this method.</p> <pre>Boolean success = CodeModuleManager.writeNewModule(handle, data, 0, data.length);</pre>
Save a module to the BlackBerry® device database.	<p>> Invoke <code>saveNewModule(int)</code>. If the module saves successfully, the method returns one of the result codes defined in the <code>CodeModuleManager</code> class.</p> <pre>int result = CodeModuleManager.saveNewModule(handle);</pre>

Task	Steps
Delete a module from the BlackBerry® device database.	<p>> Invoke <code>deleteModuleEx(int, Boolean)</code> and provide the following parameters:</p> <ul style="list-style-type: none"> the handle of the module to delete a Boolean value to specify whether to delete the module and any data it contains or to delete the module only if it does not have data associated with it. <pre>int handle = CodeModuleManager.getModuleHandle("test_module"); if(handle != 0) { Boolean success = CodeModuleManager.deleteModule(handle, true); }</pre> <p>If the module is in use, deletes it when the BlackBerry device restarts.</p>

The runtime store

BlackBerry® devices use a runtime store as a central location in which BlackBerry Java® Applications can share runtime objects. By default, only BlackBerry Java Applications that Research In Motion (RIM) digitally signs can access data in the runtime store. Contact RIM for information on how to control access to your data.

The runtime store is not persistent. A BlackBerry device restart clears the data in the runtime store.

Share runtime objects

Task	Steps
Retrieve the runtime store.	<p>> Invoke <code>RuntimeStore.getRuntimeStore()</code>.</p> <pre>RuntimeStore store = RuntimeStore.getRuntimeStore();</pre>
Add a runtime object.	<ol style="list-style-type: none"> Invoke <code>RuntimeStore.put(long, String)</code> and provide as parameters a unique long ID and the runtime object to store. Create a try - catch block to manage the <code>IllegalArgumentException</code> that <code>put()</code> throws if a runtime object with the same ID exists. <pre>RuntimeStore store = RuntimeStore.getRuntimeStore(); // Create an object and a unique number to identify the object. String msg = "Some shared text"; long ID = 0x60ac754bc0867248L; try { store.put(ID, msg); } catch(IllegalArgumentException e) { // Handle exception - an object with the same ID exists. }</pre>

Task	Steps
Replace a runtime object.	<ol style="list-style-type: none"> 1. Invoke <code>replace()</code>. 2. Create a try - catch block to manage the <code>ControlledAccessException</code> that <code>replace()</code> throws if the runtime object with the specified ID does not exist. <pre> RuntimeStore store = RuntimeStore.getRuntimeStore(); String newmsg = "Some new text"; try { Object obj = store.replace(0x60ac754bc0867248L, newmsg); } catch(ControlledAccessException e) { // Handle exception - insufficient permissions. } not exist.</pre>
Retrieve a registered runtime object.	<ol style="list-style-type: none"> 1. Invoke <code>RuntimeStore.get()</code> and provide as a parameter the runtime object ID. 2. Create a try - catch block to manage the <code>ControlledAccessException</code> that <code>get()</code> throws if the BlackBerry® Java® Application does not have read access to the specified runtime object. <pre> RuntimeStore store = RuntimeStore.getRuntimeStore(); try { // get() returns the object with the specified ID if it exists; null // otherwise. Object obj = store.get(0x60ac754bc0867248L); } catch(ControlledAccessException e) { // Handle exception. }</pre>
Retrieve an unregistered runtime object.	<ol style="list-style-type: none"> 1. Invoke <code>RuntimeStore.waitFor()</code> to wait for registration of a runtime object to complete. If the runtime object with the specified ID does not exist, <code>waitFor()</code> blocks for a maximum of <code>MAX_WAIT_MILLIS</code>. 2. Create code for handling exceptions. <pre> RuntimeStore store = RuntimeStore.getRuntimeStore(); try { Object obj = store.waitFor(0x60ac754bc0867248L); } catch(ControlledAccessException e) { // Handle exception - insufficient permissions. } catch(RuntimeException e) { // Handle exception - time out. }</pre>

Code sample

Code sample: Listening for changes to IT policies

Example: `ITPolicyDemo.java`

```

/**
 * ITPolicyDemo.java
 * Copyright (C) 2002-2005 Research In Motion Limited.
 */
```

```

package com.rim.samples.docs.itpolicy;

import net.rim.device.api.system.*;
import net.rim.device.api.itpolicy.*;

public class ITPolicyDemo extends Application implements GlobalEventListener {
    public static void main(String[] args) {
        ITPolicyDemo app = new ITPolicyDemo();
        app.enterEventDispatcher();
    }
    ITPolicyDemo() {
        this.addGlobalEventListener(this);
        boolean appEnabled = ITPolicy.getBoolean("DemoAppEnabled", true);
        System.out.println("App Enabled: " + appEnabled);
        System.exit(0);
    }
    public void eventOccurred(long guid, int data0, int data1, Object obj0, Object obj1) {
        if (guid == ITPolicy.GUID_IT_POLICY_CHANGED ) {
            String security = ITPolicy.getString("DemoSecurityLevel");
            boolean appEnabled = ITPolicy.getBoolean("DemoAppEnabled", true);
            int retries = ITPolicy.getInteger("DemoAppRetries", 10);
        }
    }
}

```

Using the messages application

Create new messages
Work with a message
Work with folders
Working with attachments

Create new messages

Task	Steps
Create a new blank text message.	<p>> Invoke <code>invokeApplication()</code> using the <code>APP_TYPE_MESSAGES</code> constant parameter and a new <code>MessageArguments</code> object that uses the <code>ARG_NEW_SMS</code> parameter.</p> <pre>Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments(MessageArguments.ARG_NEW_SMS));</pre>
Create a new populated text message.	<p>Use the API items in the <code>javax.wireless.messaging</code> package (JSR 120).</p> <ol style="list-style-type: none"> 1. Create and populate a new <code>TextMessage</code> object. <pre>MessageConnection mc = (MessageConnection)Connector.open("sms://"); TextMessage m = (TextMessage)mc.newMessage(MessageConnection.TEXT_MESSAGE); m.setAddress("sms://5558888"); m.setPayloadText("An SMS Message for you");</pre> 2. Invoke <code>invokeApplication()</code> with the following parameters: <ul style="list-style-type: none"> • <code>APP_TYPE_MESSAGES</code>: a constant parameter • <code>MessageArguments</code>: a new <code>MessageArguments</code> object that uses the new <code>TextMessage</code> object. <pre>Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments(m));</pre>
Create a new text message with multimedia.	<p>> Invoke <code>invokeApplication()</code> using the <code>APP_TYPE_MESSAGES</code> constant parameter and a new <code>MessageArguments</code> object that uses the <code>ARG_NEW_MMS</code> parameter.</p> <pre>Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments(MessageArguments.ARG_NEW_MMS));</pre>
Create a new blank email message.	<p>> Invoke <code>invokeApplication()</code> using the <code>APP_TYPE_MESSAGES</code> constant parameter and a new <code>MessageArguments</code> object that uses the <code>ARG_NEW</code> parameter.</p> <pre>Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments(MessageArguments.ARG_NEW));</pre>

Task	Steps
Create a new populated email message.	<ol style="list-style-type: none"> 1. Create and populate a new email message object. <pre> net.rim.blackberry.api.mail.Message m = new net.rim.blackberry.api.mail.Message(); Address a = new Address("mLi@rim.com", "Ming Li"); Address[] addresses = {a}; m.addRecipients(net.rim.blackberry.api.mail.Message.RecipientType.TO, addresses); m.setContent("A message for you..."); m.setSubject("Email for you"); </pre> 2. Invoke <code>invokeApplication()</code> with the following parameters: <ul style="list-style-type: none"> • <code>APP_TYPE_MESSAGES</code>: a constant parameter • <code>MessageArguments</code>: a new <code>MessageArguments</code> object that uses the new email <code>Message</code> object. <pre> Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments(m)); </pre>
Create a new blank PIN message.	<p>> Invoke <code>invokeApplication()</code> using the <code>APP_TYPE_MESSAGES</code> constant parameter and a new <code>MessageArguments</code> object that uses the <code>ARG_NEW_PIN</code> parameter.</p> <pre> Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments(MessageArguments.ARG_NEW_PIN)); </pre>
Create a new populated PIN message.	<ol style="list-style-type: none"> 1. Create and populate a new PIN message. <pre> net.rim.blackberry.api.mail.Message m = new net.rim.blackberry.api.mail.Message(); PINAddress pa = new PINAddress("ABCDEF99", "Mark Chapters"); Address[] addresses = {pa}; m.addRecipients(net.rim.blackberry.api.mail.Message.RecipientType.TO, addresses); m.setContent("A message for you..."); m.setSubject("PIN message for you"); </pre> 2. Invoke <code>invokeApplication()</code> with the following parameters: <ul style="list-style-type: none"> • <code>APP_TYPE_MESSAGES</code>: a constant parameter • <code>MessageArguments</code>: a new <code>MessageArguments</code> object that uses the new PIN message. <pre> Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments(m)); </pre>

Work with a message

Task	Steps
Receive a message notification.	<ol style="list-style-type: none"> 1. Implement the <code>FolderListener</code> and <code>StoreListener</code> interfaces. <pre>public class MailTest implements FolderListener, StoreListener { ... }</pre> 2. Create code to manage a <code>ControlledAccessException</code>.
Add a listener to the message store.	<ol style="list-style-type: none"> 1. Retrieve the <code>Store</code> object. 2. Add a <code>StoreListener</code> instance to it. 3. Create a try-catch block to manage a <code>NoSuchServiceException</code>. <pre>try { Store store = Session.waitForDefaultSession().getStore(); } catch (NoSuchServiceException e) { System.out.println(e.toString()); } store.addStoreListener(this);</pre>
Add a listener to the message store for batch updates.	<pre>> Implement StoreListener.batchOperation(). void batchOperation(StoreEvent e) { // Perform action when messages added or removed in batch operation. }</pre>
Add a listener to a folder.	<ol style="list-style-type: none"> 1. Retrieve the <code>Folder</code> object for which you want to receive new message notifications. <pre>Folder[] folders = store.list(Folder.INBOX); Folder inbox = folders[0];</pre> 2. Add the <code>FolderListener</code> instance to the folder. <pre>inbox.addFolderListener(this);</pre> 3. Implement <code>FolderListener.messagesAdded()</code> and <code>FolderListener.messagesRemoved()</code>. <pre>void messagesAdded(FolderEvent e) { // Perform processing on added messages. } void messagesRemoved(FolderEvent e) { // Perform processing on removed messages. }</pre>
Retrieve the total count of unread messages in all folders in the store.	<pre>> Invoke net.rim.blackberry.api.mail.Store.getUnreadMessageCount(). int numUnread = store.getUnreadMessageCount();</pre>

Task	Steps
Get more of a message.	<p>By default, the first section of a message (typically about 2 KB) is sent to the BlackBerry® device.</p> <ol style="list-style-type: none"> 1. Create an instance of a subclass of the <code>BodyPart</code> abstract class. <code>TextBodyPart tb = new TextBodyPart(new MultiPart());</code> 2. To determine if more data for a body part is available on the server, invoke <code>tb.hasMore()</code>. 3. To determine if the BlackBerry device user made a request for more data, invoke <code>tb.moreRequestSent()</code>. 4. To obtain a <code>Transport</code> object, invoke <code>Session.getTransport()</code> and store the returned object in a variable of type <code>Transport</code>. <code>Transport trans = Session.getTransport();</code> 5. To request more of a message, invoke <code>trans.more(BodyPart bp, boolean reqAll)</code>. The second parameter of <code>more()</code> is a Boolean value that specifies whether to retrieve only the next section of the body part (false) or all remaining sections of the body part (true). <pre>if ((tb.hasMore()) && (! tb.moreRequestSent())) {trans.more(tb, true);}</pre>

Open a message

1. Retrieve the message store and the folder that contains the message.

```
Store store = Session.waitForDefaultSession().getStore();
Folder folder = Store.getFolder("SampleFolder");
```

2. Retrieve the message objects from the folder. Iterate through the array and retrieve information, such as the sender and subject, to display to the BlackBerry® device user.

```
Message[] msgs = folder.getMessage();
```

3. When a BlackBerry device user selects a message from the list, invoke methods on the Message object to retrieve the appropriate fields and body contents to display to the BlackBerry device user.

```
Message msg = msgs[0]; // Retrieve the first message.
Address[] recipients = msg.getRecipients(Message.RecipientType.TO);
Date sent = msg.getSentDate();
Address from = msg.getFrom();
String subject = msg.getSubject();
Object o = msg.getContent();
// Verify that the message is not multipart.
if ( o instanceof String ) {
    String body = (String)o;} //...
```

4. Invoke `getBodyText()` on a message to retrieve the plain text contents as a `String`. If the message does not contain plain text, the method returns null.

Send a message

Task	Steps
Create a message.	<ol style="list-style-type: none"> 1. Create a Message object. 2. Specify a folder in which to save a copy of the sent message. <pre>Store store = Session.getDefaultInstance().getStore(); Folder[] folders = store.list(Folder.SENT); Folder sentfolder = folders[0]; Message msg = new Message(sentfolder);</pre>
Specify the recipients.	<ol style="list-style-type: none"> 1. Create an array of Address objects. 2. Add each address to the array. 3. Create code to catch an AddressException, which is thrown if an address is invalid. <pre>Address toList[] = new Address[1]; try { toList[0]= new Address("aisha.wahl@blackberry.com", "Aisha Wahl"); } catch(AddressException e) { System.out.println(e.toString()); }</pre>

Task	Steps
Add the recipients.	<ol style="list-style-type: none"> 1. Invoke <code>Message.addRecipients()</code> and provide the type of recipient (TO, CC, or BCC) and the array of addresses to add as parameters to the method. 2. If the message has multiple types of recipients, invoke <code>addRecipients()</code> once for each recipient type. <pre>msg.addRecipients(Message.RecipientType.TO, toList);</pre>
Specify the name and email address of the sender.	<pre>> Invoke setFrom(Address). Address from = new Address("scott.mcperson@blackberry.com", "Scott McPherson"); msg.setFrom(from);</pre>
Add a subject line.	<pre>> Invoke setSubject(String). msg.setSubject("Test Message");</pre>
Specify the message contents.	<pre>> Invoke setContent(String). Typically, the BlackBerry® Java® Application retrieves content from text that a BlackBerry device user types in a field. try { msg.setContent("This is a test message."); } catch (MessagingException e) { System.out.println(e.getMessage()); }</pre>
Send the message.	<ol style="list-style-type: none"> 1. Invoke <code>Session.getTransport()</code> and store the returned object in a variable of type <code>Transport</code>. The <code>Transport</code> object represents the messaging transport protocol. <pre>Transport trans = Session.getTransport();</pre> 2. Invoke <code>trans.send(Message)</code>. <pre>try { trans.send(msg); } catch (MessagingException e) { System.out.println(e.getMessage()); }</pre>

Reply to a message

Task	Steps
Reply to an existing message.	<ol style="list-style-type: none"> 1. Invoke <code>Session.getTransport()</code> and store the returned object in a variable of type <code>Transport</code>. The <code>Transport</code> object represents the messaging transport protocol. <pre>Transport trans = Session.getTransport();</pre> <pre>> Invoke Message.reply(Boolean) and specify true to reply to all message recipients or false to reply to only the sender. Store store = Session.waitForDefaultSession().getStore(); Folder[] folders = store.list(INBOX); Folder inbox = folders[0]; Message[] messages = inbox.getMessage(); if(messages.length > 0) { Message msg = messages[0]; } Message reply = msg.reply(true); trans.send(reply);</pre>

Forward a message

Task	Steps
Create a message object.	<ul style="list-style-type: none"> > Invoke <code>forward()</code> on an existing <code>Message</code> object. The subject line of a forwarded message is set automatically to <code>FW:original_subject</code>. <pre>Message fwdmsg = msg.forward();</pre>
Add the recipients.	<ol style="list-style-type: none"> 1. Create an array of addresses. <pre>Address toList[] = new Address[1];</pre> 2. Invoke <code>addRecipients(int, Address[])</code>. <pre>toList[0] = new Address("aisha.wahl@blackberry.com", "Katie Laird"); fwdmsg.addRecipients(Message.RecipientType.TO, toList);</pre>
Specify that the message content appears before the original message.	<ul style="list-style-type: none"> > Invoke <code>setContent(String)</code>. <pre>try { fwdmsg.setContent("This is a forwarded message."); } catch (MessagingException e) { System.out.println(e.getMessage()); }</pre>
Send the message.	<ol style="list-style-type: none"> 1. Invoke <code>Session.getTransport()</code> and store the returned object in a variable of type <code>Transport</code>. The <code>Transport</code> object represents the messaging transport protocol. <pre>Transport trans = Session.getTransport();</pre> 2. Invoke <code>trans.send(Message)</code>. <pre>try { trans.send(fwdmsg); } catch (MessagingException e) { System.out.println(e.getMessage()); }</pre>

Work with folders

1. Invoke `getStore()` on the default session.

```
Store store = Session.waitForDefaultSession().getStore();
```
2. Complete any of the following actions:

Task	Steps
Open a folder view.	<ol style="list-style-type: none"> 1. Invoke <code>store.list()</code> to retrieve a list of folders. <pre>Store store = null; store = Session.waitForDefaultSession().getStore(); Folder[] folders = store.list();</pre> 2. Invoke <code>invokeApplication()</code> using the <code>APP_TYPE_MESSAGES</code> constant parameter and a new <code>MessageArguments</code> object that uses a folder from the list of folders as a parameter. <pre>Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments(folders[0]));</pre>
List the folders in a mailbox store.	<ul style="list-style-type: none"> > Invoke <code>Store.list()</code>. <pre>Folder[] folders = store.list();</pre>

Task	Steps
Retrieve an array of folders by type.	<ul style="list-style-type: none"> > Invoke <code>list(int)</code> and provide as a parameter the folder type. <pre>Folder[] folders = store.list(INBOX); Folder inbox = folders[0];</pre>
Retrieve an array of folders through a search.	<ul style="list-style-type: none"> > Invoke <code>findFolder(String)</code>. <pre>Folder[] folders = store.findFolder("Inbox");</pre>
Retrieve a folder by its name.	<ol style="list-style-type: none"> 1. Invoke <code>getFolder(String)</code> and provide as a parameter the absolute path to the folder. <pre>Folder folder = store.getFolder("Mailbox - Aisha Wahl/Inbox/Projects");</pre> <ol style="list-style-type: none"> 2. Create code to manage a <code>FolderNotFoundException</code> exception if the folder does not exist.
Retrieve a folder by its ID.	<ol style="list-style-type: none"> 1. Invoke <code>getID()</code> to retrieve the folder ID. 2. Invoke <code>getFolder()</code> with the ID as a parameter. <pre>Folder[] folders = store.list(); long id = folders[0].getId(); Folder f2 = store.getFolder(id);</pre>
File a message.	<ul style="list-style-type: none"> > Invoke <code>appendMessage(Message)</code> on a <code>Folder</code> object. <pre>Message msg = new Message(); //... Folder folder = store.getFolder("Inbox"); folder.appendMessage(msg);</pre>

Working with attachments

To open incoming message attachments and create outgoing attachments on the BlackBerry® device, use the mail API. A separate `BodyPart` on a `Multipart` message represents a message attachment.

Create an attachment handler

The BlackBerry® Enterprise Server Attachment Service receives all attachments first. Third-party attachment handlers cannot override the default BlackBerry device behavior. See the *BlackBerry Enterprise Server Maintenance and Troubleshooting Guide* for more information about the BlackBerry Enterprise Server Attachment Service.

Task	Steps
Define a custom attachment handler.	> Implement the <code>AttachmentHandler</code> interface.
Register the accepted MIME types when the BlackBerry® device receives an attachment.	> Implement <code>supports(String)</code> . <pre>public boolean supports(String contentType) { return (contentType.toLowerCase().indexOf("contenttype") != -1 ? true : false); }</pre>
Define the associated menu item string to display in the messages list when the BlackBerry® device user selects an attachment.	> Implement <code>menuString()</code> . <pre>public String menuString() { return "Custom Attachment Viewer"; }</pre>
Define attachment processing.	> Implement <code>run()</code> . When a BlackBerry® device user selects a menu item from the messages list, this action invokes the <code>run()</code> method. <pre>public void run(Message m, SupportedAttachmentPart p) { // Perform processing on data. Screen view = new Screen(); view.setTitle(new LabelField("Attachment Viewer")); view.add(new RichTextField(new String((byte[])p.getContent()))); }</pre>
Register an attachment.	When registering a custom attachment handler, the attachment name must be prefixed with "x-rimdevice" for the attachment to be sent and stored on the BlackBerry® device. > Invoke <code>AttachmentHandlerManager.addAttachmentHandler()</code> . <pre>AttachmentHandlerManager m = AttachmentHandlerManager.getInstance(); CustomAttachmentHandler ah = new CustomAttachmentHandler(); m.addAttachmentHandler(ah);</pre>

Retrieve attachments

Task	Steps
Retrieve the contents of an attachment.	> Invoke <code>SupportedAttachmentPart.getContent()</code> . <pre>String s = new String((byte[])p.getContent());</pre>

Task	Steps
Retrieve information about the attachment.	<ul style="list-style-type: none"> > Invoke the methods of the <code>SupportedAttachmentPart</code> class. The <code>SupportedAttachmentPart</code> class represents an attachment with a corresponding viewer on the BlackBerry® device. An <code>UnsupportedAttachmentPart</code> represents an attachment that does not have a viewer on the BlackBerry device.

Send a message with an attachment

Task	Steps
Create a multipart message.	<ul style="list-style-type: none"> > Create a new <code>Multipart</code> object. <pre>byte[] data = new byte[256]; // The attachment. MultiPart multipart = new MultiPart(); // Default type of multipart/mixed.</pre>
Create each component of the attachment.	<ul style="list-style-type: none"> > Create a <code>SupportedAttachmentPart</code> object, designating the <code>Multipart</code> object as its parent. <pre>SupportedAttachmentPart attach = new SupportedAttachmentPart(multipart, "application/x-example", "filename", data);</pre>
Add each <code>SupportedAttachmentPart</code> object to the multipart object.	<ul style="list-style-type: none"> > Invoke <code>addBodyPart(SupportedAttachmentPart)</code> on that object. <pre>multipart.addBodyPart(attach); // Add the attachment to the multipart.</pre>
Set the content of the attachment.	<ul style="list-style-type: none"> > Invoke <code>setContent(Multipart)</code> on the <code>Message</code> object and provide as a parameter the <code>Multipart</code> object. <pre>msg.setContent(multipart);</pre>
Send the message.	<ol style="list-style-type: none"> 1. Invoke <code>Session.getTransport()</code> and store the returned object in a variable of type <code>Transport</code>. The <code>Transport</code> object represents the messaging transport protocol. <pre>Transport trans = Session.getTransport();</pre> 2. Invoke <code>trans.send(Message)</code>. <pre>try { trans.send(msg); } catch(MessagingException e) { System.out.println(e.getMessage()); }</pre>

Using PIM applications

Using the calendar
Using the address book
Using tasks
Code samples

Using the calendar

Start the calendar from your BlackBerry Java Application

Task	Steps
Open the calendar.	> Invoke <code>Invoke.invokeApplication(APP_TYPE_CALENDAR, CalendarArguments).</code>
View or change an event.	<ol style="list-style-type: none"> 1. Retrieve an Event from the list of events. <pre>Event e = null; EventList el = (EventList)PIM.getInstance().openPIMList(PIM.EVENT_LIST, PIM.READ_WRITE); Enumeration events = el.items(); e = (Event)events.nextElement();</pre> 2. Invoke <code>Invoke.invokeApplication(APP_TYPE_CALENDAR, CalendarArguments)</code> using the <code>CalendarArguments</code> object created using the <code>ARG_VIEW_DEFAULT</code> property and the retrieved Event. <pre>Invoke.invokeApplication(Invoke.APP_TYPE_CALENDAR, new CalendarArguments(CalendarArguments.ARG_VIEW_DEFAULT, e));</pre>
Manage exceptions	> Check for a <code>ControlledAccessException</code> if your BlackBerry® Java® Application invokes a BlackBerry application that you do not have permission to use or access.

Task	Steps
Open a new populated event.	<ol style="list-style-type: none"> 1. Create a new Event using an EventList object. <pre>Event e = null; EventList el = (EventList)PIM.getInstance().openPIMList(PIM.EVENT_LIST, PIM.READ_WRITE); e = el.createEvent();</pre> 2. Add information to the Event object. <pre>e.addString(Event.SUMMARY, 0, "Go For A Walk"); e.addString(Event.LOCATION, 0, "The Park"); long start = System.currentTimeMillis() + 8640000; e.addDate(Event.START, 0, start); e.addDate(Event.END, 0, start + 72000000);</pre> 3. Invoke <code>Invoke.invokeApplication(APP_TYPE_CALENDAR, CalendarArguments)</code> using the <code>CalendarArguments</code> object created using the <code>ARG_NEW</code> property and the Event. <pre>Invoke.invokeApplication(Invoke.APP_TYPE_CALENDAR, new CalendarArguments(CalendarArguments.ARG_NEW, e));</pre> 4. Use an instance of the EventList class to access the calendar. 5. Create one or more Event objects to store information for specific appointments. For each event, you can store data such as the summary, location, start and end times, and reminder notification.

Use the calendar

Task	Steps
Open an event list.	<ul style="list-style-type: none"> > Create an EventList object by invoking <code>openPIMList()</code>, providing as parameters the type of list to open (<code>PIM.EVENT_LIST</code>) and the mode in which to open the list: <ul style="list-style-type: none"> • <code>READ_WRITE</code> • <code>READ_ONLY</code> • <code>WRITE_ONLY</code> <pre>EventList eventList = null; try { eventList = (EventList)PIM.getInstance().openPIMList(PIM.EVENT_LIST, PIM.READ_WRITE); } catch (PimException e) { // Handle exception. }</pre>
Create an appointment.	<ul style="list-style-type: none"> > Invoke <code>createEvent()</code> on an event list. <pre>Event event = eventList.createEvent();</pre>

Task	Steps
Add appointment information.	<pre> > To verify that an item supports a field, invoke isSupportedField(int). if (event.isSupportedField(Event.SUMMARY)) { event.addString(Event.SUMMARY, Event.ATTR_NONE, "Meet with customer"); } if (event.isSupportedField(Event.LOCATION)) { event.addString(Event.LOCATION, Event.ATTR_NONE, "Conference Center"); } Date start = new Date(System.currentTimeMillis() + 8640000); if (event.isSupportedField(Event.START)) { event.addDate(Event.START, Event.ATTR_NONE, start); } if (event.isSupportedField(Event.END)) { event.addDate(Event.END, Event.ATTR_NONE, start + 72000000); } if (event.isSupportedField(Event.ALARM)) { if (event.countValues(Event.ALARM) > 0) { event.removeValue(Event.ALARM, 0); event.setInt(Event.ALARM, 0, Event.ATTR_NONE, 396000); } } </pre>
Create a recurring appointment.	<ol style="list-style-type: none"> 1. Create a RepeatRule object. The RepeatRule class defines fields for the properties and values that you can set, such as COUNT, FREQUENCY, and INTERVAL. 2. To retrieve an array of supported fields, invoke RepeatRule.getFields(). 3. To define a recurring pattern, invoke setInt(int, int) or setDate(int, int, int, long) on a new RepeatRule object. <pre> RepeatRule recurring = new RepeatRule(); recurring.setInt(RepeatRule.FREQUENCY, RepeatRule.MONTHLY); recurring.setInt(RepeatRule.DAY_IN_MONTH, 14); </pre> 4. To assign a recurrence pattern to an appointment, invoke setRepeat(RepeatRule) on an event. <pre> EventList eventList = (EventList)PIM.getInstance().openPIMList(PIM.EVENT_LIST, PIM.READ_WRITE); Event event = eventList.createEvent(); event.setRepeat(recurring); </pre>
Change appointment information.	<ol style="list-style-type: none"> 1. To replace an existing value with a new one, invoke the appropriate set method, such as setString(). 2. To determine if a value is already set for the field, invoke countValues(). 3. To change an existing value, use the corresponding set method, such as setString(). <pre> if (event.countValues(Event.LOCATION) > 0) { event.setString(Event.LOCATION, 0, Event.ATTR_NONE, "Board Room"); } </pre>

Task	Steps
Save an appointment.	<p>To save an appointment, use the <code>importEvent()</code> method; you do not have to invoke <code>commit()</code>.</p> <ol style="list-style-type: none"> Before you save the appointment, to identify appointment fields that have changed since the appointment was last saved, invoke <code>isModified()</code>. Invoke <code>commit()</code>. <pre>if(event.isModified()) { event.commit(); }</pre>
Retrieve appointment information.	<ol style="list-style-type: none"> To retrieve an enumeration of appointments, invoke <code>PIMList.items()</code>. <pre>EventList eventList = (EventList)PIM.getInstance().openPIMList(PIM.EVENT_LIST, PIM.READ_ONLY); Enumeration e = eventList.items();</pre> To retrieve an array of IDs of fields that have data for a particular task, invoke <code>PIMItem.getFields()</code>. To retrieve the field values, invoke <code>PIMItem.getString()</code>. <pre>while (e.hasMoreElements()) { Event event = (Event)e.nextElement(); int[] fieldIds = event.getFields(); int id; for(int index = 0; index < fieldIds.length; ++index) { id = fieldIds[index]; if(e.getPIMList().getFieldDataType(id) == STRING) { for(int j=0; j < event.countValues(id); ++j) { String value = event.getString(id, j); System.out.println(event.getFieldLabel(id) + "=" + value); } } } }</pre>
Export an appointment.	<ol style="list-style-type: none"> To import or export PIM item data, use an output stream writer to export tasks from the BlackBerry® device to a supported serial format, such as iCal®. To retrieve a string array of supported serial formats, invoke <code>PIM.supportedSerialFormats()</code>, and then specify the list type (<code>PIM.EVENT_List</code>). To write an item in serial format, invoke <code>toSerialFormat()</code>. The <code>enc</code> parameter specifies the character encoding to use when writing to the output stream. Supported character encodings include "UTF8," "ISO-8859-1," and "UTF-16BE". This parameter cannot be null. <pre>EventList eventList = (EventList)PIM.getInstance().openPIMList(PIM.EVENT_LIST, PIM.READ_ONLY); ByteArrayOutputStream bytestream = new ByteArrayOutputStream(); Enumeration e = eventList.items(); while (e.hasMoreElements()) { Event event = (Event)e.nextElement(); PIM.getInstance().toSerialFormat(event, bytestream, "UTF8", dataFormats[0]); }</pre>

Task	Steps
Import an appointment.	<ol style="list-style-type: none"> 1. To return an array of PIMItem objects, invoke from <code>SerialFormat(java.io.InputStream is, java.lang.String enc)</code>. 2. To add a new appointment, invoke <code>EventList.importEvent()</code>. // Convert an existing appointment into a iCal and then import the iCal as a new // appointment. String[] dataFormats = PIM.eventSerialFormats(); // Write appointment to iCal. ByteArrayOutputStream os = new ByteArrayOutputStream(); PIM.getInstance().toSerialFormat(event, os, "UTF8", dataFormats[0]); // Import appointment from iCal. ByteArrayInputStream is = new ByteArrayInputStream(outputStream.toByteArray()); PIMItem[] pi = PIM.getInstance().fromSerialFormat(is, "UTF8"); EventList eventList = (EventList)PIM.getInstance().openPIMList(PIM.EVENT_LIST, PIM.READ_WRITE); Event event2 = eventList.importEvent((Event)pi[0]);
Close an event list.	<ol style="list-style-type: none"> 1. Invoke <code>close()</code>. 2. Create a try-catch block to manage a <code>PimException</code>. <pre>try { eventList.close(); } catch (PimException e) { // Handle exception. }</pre>

See "Code sample: Creating new recurring appointments" on page 182 for more information.

Using the address book

Open the address book from your BlackBerry Java Application

Task	Steps
Open the address book.	<ul style="list-style-type: none"> > From a BlackBerry® Java® Application, invoke <code>Invoke.invokeApplication(APP_TYPE_ADDRESSBOOK, AddressBookArguments)</code>.
Open a contact using PIM data.	<ol style="list-style-type: none"> 1. Create an instance of an <code>AddressBookArguments</code> object, specifying as a parameter a <code>Contact</code> object. <pre>AddressBookArguments abArg = AddressBookArguments(String arg, Contact contact);</pre> 2. Invoke <code>Invoke.invokeApplication(APP_TYPE_ADDRESSBOOK, AddressBookArguments)</code> using the <code>AddressBookArguments</code> object for the contact. <pre>Invoke.invokeApplication(APP_TYPE_ADDRESSBOOK, abArg);</pre>
Manage exceptions.	<ul style="list-style-type: none"> > Check for a <code>ControlledAccessException</code> if your BlackBerry® Java® Application invokes a BlackBerry application that you do not have permission to use or access.

Use contacts

Task	Steps
Provide access to the PIN BlackBerry® device contacts field.	<ul style="list-style-type: none"> > Use the <code>BlackBerryContact.PIN</code> constant.
Provide access to the USER1 through USER4 BlackBerry® device contacts fields.	<ul style="list-style-type: none"> > Use the following constants: <ul style="list-style-type: none"> • <code>BlackBerryContact.USER1</code> • <code>BlackBerryContact.USER2</code> • <code>BlackBerryContact.USER3</code> • <code>BlackBerryContact.USER4</code>
Define labels for the USER1 through USER4 BlackBerry® device contacts fields.	<p>Changing a label affects all contacts on the BlackBerry device.</p> <ul style="list-style-type: none"> > Invoke <code>BlackBerryPIMList.setFieldLabel()</code>.
Open a contacts list.	<ol style="list-style-type: none"> 1. Create a contacts list. <pre>ContactList contactList = null;</pre> 2. Invoke <code>PIM.openPIMList()</code> and provide as parameters the type of list to open (<code>PIM.CONTACT_LIST</code>) and the access mode with which to open the list (<code>READ_WRITE</code>, <code>READ_ONLY</code>, or <code>WRITE_ONLY</code>). <pre>try { contactList = (ContactList)PIM.getInstance().openPIMList(PIM.CONTACT_LIST, PIM.READ_WRITE); } catch (PimException e) { return; }</pre>

Task	Steps
Create a contact.	<p>To add a contact to the database, you must commit it. See “Save a contact” on page 173 for more information about committing contact data.</p> <ul style="list-style-type: none">> Invoke <code>createContact()</code> on a contact list. <pre>Contact contact = contactList.createContact();</pre>

Task	Steps
Add contact information.	<ol style="list-style-type: none"> 1. Invoke one of the following methods: <ul style="list-style-type: none"> • <code>addString()</code> • <code>addStringArray()</code> • <code>addDate()</code> • <code>addInt()</code> • <code>addBoolean()</code> • <code>addBinary()</code> 2. Before you set or retrieve a field, to verify that the item supports the field, invoke <code>ContactList.isSupportedField(int)</code>. 3. To let fields store multiple values, use field attributes. For example, the TEL field supports the ATTR_HOME, ATTR_WORK, ATTR_MOBILE, and ATTR_FAX attributes to store numbers for work, home, mobile, and fax numbers. 4. To determine how many values a field supports, invoke <code>PIMList.maxValues(int field)</code>. 5. To verify that a field supports a particular attribute, invoke <code>isSupportedAttribute(int, int)</code>. <pre>// Create string array for name. try {ContactList contactList = (ContactList)PIM.getInstance().openPIMList(PIM.CONTACT_LIST, PIM.WRITE_ONLY);} catch (PIMException e) {} Contact contact = contactList.createContact();String[] name = new String[contactList.stringArraySize(Contact.NAME)]; // 5 name elements try {name[Contact.NAME_PREFIX] = "Mr.";name[Contact.NAME_FAMILY] = "McPherson";name[Contact.NAME_GIVEN] = "Scott";} catch (IllegalArgumentException iae) {// handle exception} // Add name. if(contactList.isSupportedField(Contact.NAME)) {contact.addStringArray(Contact.NAME, Contact.ATTR_NONE, name); } // Create string array for address. String[] address = new String[7]; // 7 address elements try {address[Contact.ADDR_COUNTRY] = "United States";address[Contact.ADDR_LOCALITY] = "Los Angeles";address[Contact.ADDR_POSTALCODE] = "632300";address[Contact.ADDR_REGION] = "California";address[Contact.ADDR_STREET] = "323 Main Street";} catch (IllegalArgumentException iae) {// Handle exception.} // Add address.contact.addStringArray(Contact.ADDR, Contact.ATTR_NONE, address); // Add home telephone number. if (contactList.isSupportedField(Contact.TEL) &&contactList.isSupportedAttribute(Contact.TEL, Contact.ATTR_HOME)) {contact.addString(Contact.TEL, Contact.ATTR_HOME, "555-1234");} // Add work telephone number.if (contactList.isSupportedField(Contact.TEL)) {contact.addString(Contact.TEL, Contact.ATTR_HOME, "555-5555");} // Add work internet messaging address. if (contactList.isSupportedField(Contact.EMAIL)) {contact.addString(Contact.EMAIL, Contact.ATTR_WORK, "aisha.wahl@blackberry.com");}</pre>

Task	Steps
Change contact information.	<ol style="list-style-type: none"> 1. To change the name and address fields, invoke the appropriate set method to replace an existing value with a new value. 2. Perform one of the following actions: <ul style="list-style-type: none"> • To change the fields that support a single value, retrieve the array and then change one or more indexes in the array before adding the array back to the <code>Contact</code> object. <pre> if (contact.countValues(Contact.NAME) > 0) { String[] newname = contact.getStringArray(Contact.NAME, 0); } // Change the prefix to Dr. and add the suffix, Jr. newname[Contact.NAME_PREFIX] = "Dr."; newname[Contact.NAME_SUFFIX] = "Jr."; contact.setStringArray(Contact.NAME, 0, Contact.ATTR_NONE, newname); </pre> • To change the contacts fields that support multiple values, before adding another value, verify that the number of values does not exceed the maximum number of values. For example: <pre> if (contact.countValues(Contact.EMAIL) < contactList.maxValues(Contact.EMAIL)) { contact.addString(Contact.EMAIL, Contact.ATTR_NONE, "aisha.wahl@blackberry.com");} </pre> 3. Create code to manage a <code>FieldFullException</code>, which occurs if you invoke an add method, such as <code>addString()</code>, for a field that already has a value.
Save a contact.	<ol style="list-style-type: none"> 1. To determine if any contact fields have changed since the contact was last saved, invoke <code>isModified()</code>. 2. Invoke <code>commit()</code>. <pre> if(contact.isModified()) { contact.commit(); } </pre>

Task	Steps
Retrieve contact information.	<ol style="list-style-type: none"> 1. Invoke <code>PIMList.items()</code>. 2. Perform one of the following actions: <ul style="list-style-type: none"> • To retrieve an array of IDs for fields that have data for a particular contact, invoke <code>PIMItem.getFields()</code>. • To retrieve the field values, invoke <code>PIMItem.getString()</code>. 3. When you invoke <code>PIMList.items()</code> to retrieve an enumeration of items in a contacts list, your BlackBerry® Java® Application must sort items as necessary. <pre> ContactList contactList = (ContactList)PIM.getInstance().openPIMList(PIM.CONTACT_LIST, PIM.READ_WRITE); Enumeration enum = contactList.items(); while (enum.hasMoreElements()) { Contact c = (Contact)enum.nextElement(); int[] fieldIds = c.getFields(); int id; for(int index = 0; index < fieldIds.length; ++index) { id = fieldIds[index]; if(c.getPIMList().getFieldDataType(id) == Contact.STRING) { for(int j=0; j < c.countValues(id); ++j) { String value = c.getString(id, j); System.out.println(c.getPIMList().getFieldLabel(id) + "=" + value); } } } } </pre>
Select a contact from the address book.	<pre> > Invoke the BlackBerryContactList.choose() to return a Contact or BlackBerryContactGroup PIMItem. BlackBerryContactList list = (BlackBerryContactList)PIM.getInstance().openPIMList(PIM.CONTACT_ LIST, PIM.READ_WRITE); PIMItem item = list.choose(); if (item instanceof Contact) { Contact contact = (Contact)item; String email = contact.getString(Contact.EMAIL, 0); System.out.println("Name is: " + email); } else if (item instanceof BlackBerryContactGroup) { ... } </pre>

Task	Steps
Export a contact.	<ol style="list-style-type: none"> 1. To import or export PIM item data, use an output stream writer to export tasks from the BlackBerry® device to a supported serial format, such as vCard®. 2. To retrieve a string array of supported formats, invoke <code>PIM.supportedSerialFormats()</code> and specify the list type (<code>PIM.Contact_LIST</code>). 3. To write an item to a supported serial format, invoke <code>toSerialFormat()</code>. The <code>enc</code> parameter specifies the character encoding to use when writing to the output stream. Supported character encodings include "UTF8," "ISO-8859-1," and "UTF-16BE." This parameter cannot be null. <pre> ContactList contactList = (ContactList)PIM.getInstance().openPIMList(PIM.CONTACT_LIST, PIM.READ_ONLY); String[] dataFormats = PIM.getInstance().supportedSerialFormats(PIM.CONTACT_LIST); ByteArrayOutputStream byteStream = new ByteArrayOutputStream(); Enumeration e = contactList.items(); while (e.hasMoreElements()) { Contact c = (Contact)e.nextElement(); PIM.getInstance().toSerialFormat(c, byteStream, "UTF8", dataFormats[0]); } </pre>
Import a contact.	<ol style="list-style-type: none"> 1. To return an array of PIM items, invoke <code>fromSerialFormat()</code>. 2. To create a new contact using the PIM item, invoke <code>ContactList.importContact()</code>. 3. To specify the character encoding to use when writing to the output stream, use the <code>enc</code> parameter. <pre> // Import contact from vCard. ByteArrayInputStream istream = new ByteArrayInputStream(outputStream.toByteArray()); PIMItem[] pi = PIM.getInstance().fromSerialFormat(istream, "UTF8"); ContactList contactList = (ContactList)PIM.getInstance().openPIMList(PIM.CONTACT_LIST, PIM.READ_WRITE); Contact contact2 = contactList.importContact((Contact)pi[0]); contact2.commit(); </pre>
Delete a contact.	<pre> > Invoke <code>removeContact()</code> on a contact list. contactList.removeContact(contact); </pre>
Close a contacts list.	<pre> > Invoke <code>close()</code>. try { contactList.close(); } catch(PIMException e) { Dialog.alert(e.toString()); } </pre>

See "Code sample: Displaying a screen that lets BlackBerry device users add new contacts" on page 184 for more information.

Using tasks

Start the task application from your BlackBerry Java Application

Check for a `ControlledAccessException` if your BlackBerry® Java® Application invokes a BlackBerry application that you do not have permission to use or access.

Task	Steps
Open the task application.	<p>The <code>TaskArguments</code> (<code>net.rim.blackberry.api.invoke.TaskArguments</code>) cannot be updated without changes to the Task application.</p> <pre>> Invoke invokeApplication(APP_TYPE_TASKS, TaskArguments);</pre>
View or change a task.	<ol style="list-style-type: none"> 1. Create an instance of a <code>ToDoList</code> and store it in an <code>Enumeration</code>. <pre>ToDoList tdl = (ToDoList)PIM.getInstance().openPIMList(PIM.TODO_LIST, PIM.READ_WRITE); Enumeration todos = tdl.items();</pre> 2. Create a <code>ToDo</code> object using an element from the <code>Enumeration</code>: <pre>ToDo todo = (ToDo)todos.nextElement();</pre> 3. Invoke <code>invokeApplication()</code> using the <code>APP_TYPE_TASKS</code> constant parameter, and a new <code>TaskArguments</code> object created using the <code>ARG_VIEW</code> parameter and the <code>ToDo</code> object. <pre>Invoke.invokeApplication(Invoke.APP_TYPE_TASKS, new TaskArguments(TaskArguments.ARG_VIEW, todo));</pre>
Create a new blank task.	<pre>> Invoke invokeApplication() using the APP_TYPE_TASKS constant parameter, and a new TaskArguments object created using the ARG_NEW parameter. Invoke.invokeApplication(Invoke.APP_TYPE_TASKS, new TaskArguments(TaskArguments.ARG_NEW));</pre>
Create a new populated task.	<ol style="list-style-type: none"> 1. Create an instance of a <code>ToDoList</code>. <pre>ToDoList tdl = (ToDoList)PIM.getInstance().openPIMList(PIM.TODO_LIST, PIM.READ_WRITE);</pre> 2. Invoke <code>createToDo()</code> to create a new <code>ToDo</code> object and add information to the new <code>ToDo</code> object. <pre>ToDo todo = tdl.createToDo(); todo.addString(ToDo.SUMMARY, 0, "Walk the Dog");</pre> 3. Invoke <code>invokeApplication()</code> using the <code>APP_TYPE_TASKS</code> constant parameter, and a new <code>TaskArguments</code> object created using the <code>ARG_NEW</code> parameter and the new <code>ToDo</code> object. <pre>Invoke.invokeApplication(Invoke.APP_TYPE_TASKS, new TaskArguments(TaskArguments.ARG_NEW, todo));</pre>

Use tasks

Task	Steps
Open a task list.	<pre>> Invoke PIM.openPIMList() and provide as parameters the type of list to open (PIM.TODO_LIST) and the access mode with which to open the list(READ_WRITE, READ_ONLY, or WRITE_ONLY). ToDoList todoList = null; try { todoList = (ToDoList)PIM.getInstance().openPIMList(PIM.TODO_LIST, PIM.READ_WRITE); } catch (PimException e) { //an error occurred return; }</pre>
Create a task.	<pre>> Invoke createToDo() on a task list. ToDo task = todoList.createToDo();</pre>
Add task information.	<pre>1. Before you set or retrieve a field, verify that the item supports the field by invoking isSupportedField(int). 2. To retrieve the field data type, invoke PIMList.getFieldDataType(int). 3. To set the field data, invoke one of the following methods: • addString() • addDate() • addInt() • addBoolean() • addBinary() if (todoList.isSupportedField(ToDo.SUMMARY)) { task.addString(ToDo.SUMMARY, ToDo.ATTR_NONE, "Create project plan"); } if (todoList.isSupportedField(ToDo.DUE)) { Date date = new Date(); task.addDate(ToDo.DUE, ToDo.ATTR_NONE, (date + 17280000)); } if (todoList.isSupportedField(ToDo.NOTE)) { task.addString(ToDo.NOTE, ToDo.ATTR_NONE, "Required for meeting"); } if (todoList.isSupportedField(ToDo.PRIORITY)) { task.addInt(ToDo.PRIORITY, ToDo.ATTR_NONE, 2); }</pre>
Set the status of a task.	<pre>> Use the PIM extended field ToDo.EXTENDED_FIELD_MIN_VALUE + 9: • STATUS_NOT_STARTED: 1 • STATUS_IN_PROGRESS: 2 • STATUS_COMPLETED: 3 • STATUS_WAITING: 4 task.addInt(ToDo.EXTENDED_FIELD_MIN_VALUE + 9, ToDo.ATTR_NONE, 2);</pre>

Task	Steps
Change task information.	<ol style="list-style-type: none"> 1. To replace an existing value with a new value, invoke the appropriate set method, such as <code>setString()</code>. 2. To determine if a value is already set for the field, invoke <code>countValues()</code>. 3. To change an existing value, use the corresponding <code>set()</code> method. 4. Create code to manage a <code>FieldFullException</code> which a method such as <code>addString()</code> throws when a value already exists. <pre> if (task.countValues(ToDo.SUMMARY) > 0) { task.setString(ToDo.SUMMARY, 0, ToDo.ATTR_NONE, "Review notes"); } </pre>
Save a task.	<ol style="list-style-type: none"> 1. Before you commit your changes, to determine whether any task fields have changed since the task was last saved, invoke <code>isModified()</code> 2. Invoke <code>commit()</code>. <pre> if(task.isModified()) { task.commit(); } </pre>
Retrieve task information.	<ol style="list-style-type: none"> 1. To retrieve an enumeration, invoke <code>PIMList.items()</code> on the task list. <pre> ToDoList todoList = (ToDoList)PIM.getInstance().openToDoList(PIM.TODO_LIST, PIM.READ_ONLY); Enumeration enum = todoList.items(); </pre> 2. To retrieve an array of IDs for fields that have data for a particular <code>ToDo</code> item, invoke <code>PIMItem.getFields()</code>. 3. To retrieve the field values, invoke <code>PIMItem.getString()</code>. <pre> while (enum.hasMoreElements()) { ToDo task = (ToDo)enum.nextElement(); int[] fieldIds = task.getFields(); int id; for(int index = 0; index < fieldIds.length; ++index) { id = fieldIds[index]; if(task.getPIMList().getFieldDataType(id) == STRING) { for(int j=0; j < task.countValues(id); ++j) { String value = task.getString(id, j); System.out.println(task.getFieldLabel(id) + "=" + value); } } } } } </pre>

Task	Steps
Export a task.	<ol style="list-style-type: none"> 1. To import or export PIM item data, use an output stream writer to export tasks from the BlackBerry® device to a supported serial format. 2. To retrieve a string array of supported serial formats, invoke <code>PIM.supportedSerialFormats()</code>, and then specify the list type (<code>PIM.TODO_List</code>). 3. To write an item to a serial format, invoke <code>toSerialFormat()</code>. The <code>enc</code> parameter specifies the character encoding to use when writing to the output stream. Supported character encodings include "UTF8," "ISO-8859-1," and "UTF-16BE." This parameter cannot be null. <pre> ToDoList todoList = (ToDoList)PIM.getInstance().openPIMList(PIM.TODO_LIST, PIM.READ_ONLY); ByteArrayOutputStream byteStream = new ByteArrayOutputStream(); String[] dataFormats = PIM.getInstance().supportedSerialFormats(PIM.TODO_LIST); Enumeration e = todoList.items(); while (e.hasMoreElements()) { ToDo task = (ToDo)e.nextElement(); PIM.getInstance().toSerialFormat(task, byteStream, "UTF8", dataFormats[0]); } </pre>
Import a task.	<ol style="list-style-type: none"> 1. To return an array of <code>PIMItem</code> objects, invoke <code>fromSerialFormat()</code>. The <code>enc</code> parameter specifies the character encoding to use when writing to the output stream. Supported character encodings include "UTF8," "ISO-8859-1," and "UTF-16BE." This parameter cannot be null. 2. To create a new task using the PIM items, invoke <code>ToDoList.importToDo()</code>. The <code>importToDo()</code> method saves the task; you do not have to invoke <code>commit()</code>. <pre> String[] dataFormats = PIM.toDoSerialFormats(); // Write task to serial format. ByteArrayOutputStream os = new ByteArrayOutputStream(); PIM.getInstance().toSerialFormat(task, os, "UTF8", dataFormats[0]); // Import task from serial format. ByteArrayInputStream is = new ByteArrayInputStream(outputStream.toByteArray()); PIMItem[] pi = PIM.getInstance().fromSerialFormat(is, "UTF8"); ToDoList todoList = (ToDoList)PIM.getInstance().openPIMList(PIM.TODO_LIST, PIM.READ_WRITE); ToDo task2 = todoList.importToDo((ToDo)pi[0]); </pre>
Delete a task.	<pre> > Invoke removeToDo() on a task list. todoList.removeToDo(task); </pre>
Close a task list.	<ol style="list-style-type: none"> 1. Invoke <code>todoList.close()</code>. 2. Create code that manages exceptions. <pre> try { todoList.close(); } catch (PimException e) { // Handle exception. } </pre>

See "Code sample: Using tasks" on page 186 for more information.

Code samples

Code sample: Creating new recurring appointments

To let the BlackBerry® device user invite attendees to the meeting, combine this sample with ContactsDemo.java. See "Code sample: Displaying a screen that lets BlackBerry device users add new contacts" on page 184 for more information.

Example: EventDemo.java

```
/**
 * EventDemo.java
 * Copyright (C) 2002-2005 Research In Motion Limited.
 */

package com.rim.samples.docs.eventdemo;
import java.io.*;
import java.util.*;
import javax.microedition.pim.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;

public final class EventDemo extends UiApplication
{
    private EventScreen _eventScreen;

    public static void main(String[] args) {
        new EventDemo().enterEventDispatcher();
    }

    private EventDemo() {
        _eventScreen = new EventScreen();
        pushScreen(_eventScreen);
    }

    public final static class EventScreen extends MainScreen
    {
        private EditField _subject, _location;
        private SaveMenuItem _saveMenuItem;
        private DateField _startTime, _endTime;
        private ObjectChoiceField _repeat;
        private Event event;

        private class SaveMenuItem extends MenuItem {
            public SaveMenuItem() {
                super(null, 0, 100000, 5);
            }
        }
    }
}
```

```

    public String toString() {
        return "Save";
    }

    public void run() {
        onSave();
    }
}

public EventScreen() {
    _saveMenuItem = new SaveMenuItem();
    setTitle(new LabelField("Event Demo", LabelField.ELLIPSIS |
        LabelField.USE_ALL_WIDTH) );
    _subject = new EditField("Subject: ", "");
    add(_subject);
    _location = new EditField("Location: ", "");
    add(_location);
    _startTime = new DateField("Start: ", System.currentTimeMillis() +
        3600000, DateField.DATE_TIME);
    _endTime = new DateField("End: ", System.currentTimeMillis() +
        7200000, DateField.DATE_TIME);
    add(new SeparatorField());
    add(_startTime);
    add(_endTime);
    add(new SeparatorField());
    String[] choices = {"None", "Daily", "Weekly", "Monthly", "Yearly"};
    _repeat = new ObjectChoiceField("Recurrence: ", choices, 0);
    add(_repeat);
}

protected boolean onSave() {
    try {
        EventList eventList = (EventList)PIM.getInstance().
            openPIMList(PIM.EVENT_LIST, PIM.WRITE_ONLY);
        event = eventList.createEvent();
        event.addString(Event.SUMMARY, PIMItem.ATTR_NONE,
            _subject.getText());
        event.addString(Event.LOCATION, PIMItem.ATTR_NONE,
            _location.getText());
        event.addDate(Event.END, PIMItem.ATTR_NONE, _endTime.getDate());
        event.addDate(Event.START, PIMItem.ATTR_NONE,
            _startTime.getDate());
        if(_repeat.getSelectedIndex() != 0) {
            event.setRepeat(setRule());
        }
        // Save the appointment to the Calendar.
        event.commit();
        //reset fields on screen
        _subject.setText("");
        _location.setText("");
        _endTime.setDate(null);
        _startTime.setDate(null);
        _repeat.setSelectedIndex(0);
        return true;
    } catch (PIMException e) {
        System.err.println(e);
    }
}

```

```

        return false;
    }

    private RepeatRule setRule() {
        RepeatRule rule = new RepeatRule();
        int index = _repeat.getSelectedIndex();
        if (index == 0) {
            rule.setInt(RepeatRule.FREQUENCY, RepeatRule.DAILY);
        }
        if (index == 1) {
            rule.setInt(RepeatRule.FREQUENCY, RepeatRule.WEEKLY);
        }
        if (index == 2) {
            rule.setInt(RepeatRule.FREQUENCY, RepeatRule.MONTHLY);
        }
        if (index == 3) {
            rule.setInt(RepeatRule.FREQUENCY, RepeatRule.YEARLY);
        }
        return rule;
    }

    protected void makeMenu(Menu menu, int instance) {
        menu.add(_saveMenuItem);
        menu.addSeparator();
        super.makeMenu(menu, instance);
    }
}

```

Code sample: Displaying a screen that lets BlackBerry device users add new contacts

The following sample demonstrates how to display a screen that lets BlackBerry® device users add new contacts to the address book.

Example: ContactsDemo.java

```

/**
 * ContactsDemo.java
 * Copyright (C) 2002-2005 Research In Motion Limited.
 */

package com.rim.samples.docs.contactsdemo;

import java.io.*;
import java.util.*;
import javax.microedition.pim.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;

```

```

import net.rim.blackberry.api.pdap.*;

public final class ContactsDemo extends UiApplication
{
    private ContactScreen _contactScreen;

    public static void main(String[] args) {
        new ContactsDemo().enterEventDispatcher();
    }

    public ContactsDemo() {
        _contactScreen = new ContactScreen();
        pushScreen(_contactScreen);
    }

    // Inner class. Creates a Screen to add a contact.
    public static final class ContactScreen extends MainScreen
    {
        private EditField _first, _last, _email, _phone, _pin;
        private SaveMenuItem _saveMenuItem;
        private class SaveMenuItem extends MenuItem {
            private SaveMenuItem() {
                super(null, 0, 100000, 5);
            }
            public String toString() {
                return "Save";
            }
            public void run() {
                onSave();
            }
        }

        public ContactScreen() {
            _saveMenuItem = new SaveMenuItem();
            setTitle(new LabelField("Contacts Demo", LabelField.ELLIPSIS |
LabelField.USE_ALL_WIDTH));
            _first = new EditField("First Name: ", "");
            add(_first);
            _last = new EditField("Last Name: ", "");
            add(_last);
            _email = new EditField("Email Address: ", "",
BasicEditField.DEFAULT_MAXCHARS, BasicEditField.FILTER_EMAIL);
            add(_email);
            _phone = new EditField("Work Phone: ", "",
BasicEditField.DEFAULT_MAXCHARS, BasicEditField.FILTER_PHONE);
            add(_phone);
            _pin = new EditField("PIN:", "", 8, BasicEditField.FILTER_HEXADECIMAL);
            add(_pin);
        }

        protected boolean onSave() {
            String firstName = _first.getText();
            String lastName = _last.getText();
            String email = _email.getText();
            String phone = _phone.getText();
            String pin = _pin.getText();
            // Verify that a first or last name and email has been entered.
            if ((firstName.equals("") && lastName.equals("")) || email.equals("")) {

```

```

        Dialog.inform("You must enter a name and an email address!");
        return false;
    } else {
        try {
            ContactList contactList =
(ContactList)PIM.getInstance().openPIMList(PIM.CONTACT_LIST, PIM.WRITE_ONLY);
            Contact contact = contactList.createContact();
            String[] name = new String[contactList.stringArraySize(Contact.NAME)];
            // Add values to PIM item.
            if (!firstName.equals("")) {
                name[Contact.NAME_GIVEN] = firstName;
            }
            if (!lastName.equals("")) {
                name[Contact.NAME_FAMILY] = lastName;
            }
            contact.addStringArray(Contact.NAME, Contact.ATTR_NONE, name);
            contact.addString(Contact.EMAIL, Contact.ATTR_HOME, email);
            contact.addString(Contact.TEL, Contact.ATTR_WORK, phone);
            if (contactList.isSupportedField(BlackBerryContact.PIN)) {
                contact.addString(BlackBerryContact.PIN, Contact.ATTR_NONE, pin);
            }
            // Save data to address book.
            contact.commit();
            // Reset UI fields.
            _first.setText("");
            _last.setText("");
            _email.setText("");
            _phone.setText("");
            _pin.setText("");
            return true;
        } catch (PIMException e) {
            return false;
        }
    }
}

protected void makeMenu(Menu menu, int instance) {
    menu.add(_saveMenuItem);
    super.makeMenu(menu, instance);
}
}

```

Code sample: Using tasks

Example: TaskDemo.java

```

/**
 * TaskDemo.java
 * Copyright (C) 2002-2005 Research In Motion Limited.
 */

package com.rim.samples.docs.taskdemo;

```

```

import java.io.*;
import java.util.*;
import javax.microedition.pim.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;

public final class TaskDemo extends UiApplication
{
    private TaskScreen _taskScreen;

    public static void main(String[] args) {
        new TaskDemo().enterEventDispatcher();
    }

    private TaskDemo() {
        _taskScreen = new TaskScreen();
        pushScreen(_taskScreen);
    }

    public final static class TaskScreen extends MainScreen
    {
        // Members.
        private EditField _summary, _note;
        private DateField _due;
        private ObjectChoiceField _priority, _status;
        private SaveMenuItem _saveMenuItem;

        private class SaveMenuItem extends MenuItem
        {
            private SaveMenuItem() {
                super(null, 0, 100000, 5);
            }

            public String toString() {
                return "Save";
            }

            public void run() {
                onSave();
            }
        }

        public TaskScreen() {
            _saveMenuItem = new SaveMenuItem();
            setTitle(new LabelField("Tasks Demo",
                LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH));
            _summary = new EditField("Task Summary: ", "");
            add(_summary);
            // In TODO.Priority, 0 to 9 is highest to lowest priority.
            String[] choices = {"High", "Normal", "Low"};
            _priority = new ObjectChoiceField("Priority: ", choices, 1);
            add(_priority);
            String[] status = {"Not Started", "In Progress", "Completed",
                "Waiting on someone else", "Deferred"};

```

```

        _status = new ObjectChoiceField("Status: ", status, 0);
        add(_status);
        _due = new DateField("Due: ", System.currentTimeMillis() + 3600000,
            DateField.DATE_TIME);
        add(_due);
        _note = new EditField("Extra Notes: ", "");
        add(_note);
    }

    protected boolean onSave() {
        try {
            ToDoList todoList = (ToDoList)PIM.getInstance().
                openPIMList(PIM.TODO_LIST, PIM.WRITE_ONLY);
            ToDo task = todoList.createToDo();
            task.addDate(ToDo.DUE, ToDo.ATTR_NONE, _due.getDate());
            task.addString(ToDo.SUMMARY, ToDo.ATTR_NONE, _summary.getText());
            task.addString(ToDo.NOTE, ToDo.ATTR_NONE, _note.getText());
            task.addInt(ToDo.PRIORITY, ToDo.ATTR_NONE,
                _priority.getSelectedIndex());
            // ToDo.EXTENDED_FIELD_MIN_VALUE + 9 represents status.
            // Add 1 to selected index so that values are correct.
            // See the RIM Implementation Notes in the API documentation for ToDo.
            task.addInt(ToDo.EXTENDED_FIELD_MIN_VALUE + 9, ToDo.ATTR_NONE,
                _status.getSelectedIndex() + 1);
            // Save task to handheld tasks.
            task.commit();
            _summary.setText("");
            _note.setText("");
            _due.setDate(null);
            _priority.setSelectedIndex(1); // Reset to "Normal" priority.
            _status.setSelectedIndex(0); // Reset to "Not Started" status.
            return true;
        } catch (PIMException e) {
            return false;
        }
    }

    protected void makeMenu(Menu menu, int instance) {
        menu.add(_saveMenuItem);
        super.makeMenu(menu, instance);
    }
}

```

Using the phone application

Start the phone application from your BlackBerry Java Application
 Use phone call functionality
 Listen for phone events
 Access and use call logs
 Code sample

Start the phone application from your BlackBerry Java Application

To open the phone application from your BlackBerry® Java® Application, invoke

```
Invoke.invokeApplication(APP_TYPE_PHONE, PhoneArguments).
```

The following excerpt from the Restaurants.java code sample on page 69 creates a menu item that invokes the phone application to call a restaurant.

```
private MenuItem phoneItem = new MenuItem(_resources.getString(MENUITEM_PHONE), 110, 12) {
public void run() {
synchronized(store) {
String phoneNumber = phonefield.getText();
if ( phoneNumber.length == 0 ) {
Dialog.alert(_resources.getString(ALERT_NO_PHONENUMBER));
} else {
PhoneArguments call = new PhoneArguments(PhoneArguments.ARG_CALL, phoneNumber);
Invoke.invokeApplication(Invoke.APP_TYPE_PHONE, call);
}
}
}
};
```

Use phone call functionality

Task	Steps
Retrieve a phone call.	<pre>> Invoke Phone.getActiveCall(). PhoneCall call = Phone.getActiveCall();</pre>
Retrieve the phone number of a BlackBerry device.	<pre>> Invoke Phone.getDevicePhoneNumber(boolean format). String phNumber = Phone.getDevicePhoneNumber(true);</pre>
Retrieve a phone call by call ID.	<pre>> Invoke Phone.getCall(int).</pre>

Task	Steps
Retrieve phone call information.	<pre>> Use the methods of the PhoneCall class. int threshold = 120; // Alert user if outgoing calls last longer than threshold. int elapsedTime = call.getElapsedTime(); // Use getStatusString() to retrieve status as an string. int status = call.getStatus(); if ((status == PhoneCall.STATUS_CONNECTED status == PhoneCall.STATUS_CONNECTING) && call.isOutGoing() && elapsedTime > threshold) { // Use getCallId() to retrieve the caller ID as as an integer. String phoneNumber = call.getDisplayPhoneNumber(); Status.show("Your call to " + phoneNumber + " has lasted more than " + (String)threshold + "."); }</pre>

Add DTMF tones to the send queue

Task	Steps
Add a single DTMF tone to the send queue.	<pre>> Invoke sendDTMFTone().</pre>
Add multiple DTMF tones to the send queue.	<pre>> Invoke sendDTMFTones().</pre>
Retrieve the send queue for the current call.	<pre>> Invoke getDTMFTones().</pre>

BlackBerry DTMF tones

BlackBerry® devices play DTMF tones as soon as no other tones are pending, overriding conversations.

DTMF tones consist of a low and a high frequency, which are played at the same time.

Key	Low Tone (Hz)	High Tone (Hz)
1	697	1209
2	697	1336
3	697	1477
4	770	1209
5	770	1336
6	770	1477
7	852	1209
8	852	1336
9	852	1477
0	941	1209
*	941	1336
#	941	1477

Listen for phone events

Task	Steps
Listen for phone events.	> Implement the <code>PhoneListener</code> interface.
Register the phone listener.	> Invoke <code>Phone.addPhoneListener()</code> .
Remove a phone listener.	> Invoke <code>removePhoneListener()</code> .

To act on a particular event, implement one of the following methods.

Event	Method
A call is added to a conference call.	<code>callAdded(int)</code>
A BlackBerry® device user answers a call (user driven).	<code>callAnswered(int)</code>
A conference call is established.	<code>callConferenceCallEstablished(int)</code>
The network indicates a connected event (network driven).	<code>callConnected(int)</code>
A direct-connect call is connected.	<code>callDirectConnectConnected(int)</code>
A direct-connect call is disconnected.	<code>callDirectConnectDisconnected(int)</code>
A call is disconnected.	<code>callDisconnected(int)</code>
A BlackBerry device user ends the call.	<code>callEndedByUser(int)</code>
A call fails.	<code>callFailed(int, int)</code>
A call goes on hold.	<code>callHeld(int)</code>
A new call arrives.	<code>callIncoming(int)</code>
The BlackBerry device initiates an outgoing call.	<code>callInitiated(int)</code>
A call is removed from a conference call.	<code>callRemoved(int)</code>
A held call resumes.	<code>callResumed(int)</code>
A call is waiting.	<code>callWaiting(int)</code>
A conference call is ended (all members are disconnected).	<code>conferenceCallDisconnected(int)</code>

Access and use call logs

Task	Steps
Retrieve a phone log.	<p>The <code>PhoneLogs</code> class represents the phone call history and provides methods for opening, adding, deleting, or swapping call logs.</p> <pre>> Invoke PhoneLogs.getInstance(); PhoneLogs _logs = PhoneLogs.getInstance();</pre>
Retrieve the number of normal or missed calls.	<p>There are two phone log folders: <code>FOLDER_NORMAL_CALLS</code> and <code>FOLDER_MISSED_CALLS</code>.</p> <pre>> Invoke numberOfCalls(int). int numberOfCalls = _logs.numberOfCalls(FOLDER_NORMAL_CALLS);</pre>
Retrieve a call log.	<p>You can instantiate two types of call logs: <code>PhoneCallLog</code> objects, which have only one participant, and <code>ConferencePhoneCallLog</code> objects, which have two or more participants.</p> <pre>> Invoke PhoneLogs.callAt(int index, long folderID). PhoneCallLog phoneLog = (PhoneCallLog)_logs.callAt(0);</pre>

Task	Steps
Retrieve a call participant by phone number.	<p>The PhoneCallLogID class identifies participants in a phone call log by phone number.</p> <p>> Invoke PhoneCallLog.getParticipant(int) or ConferencePhoneCallLog.getParticipantAt().</p> <pre>PhoneCallLogID participant = phoneCallLog.getParticipant(); PhoneCallLogID participant = ConferencePhoneCallLog.getParticipant();</pre>
Retrieve the phone number type.	<p>The PhoneCallLogID class identifies the type of phone call for a log. For example, home, mobile, work, or fax, as recorded in the address book.</p> <p>> Invoke PhoneCallLogID.getType().</p> <pre>String phoneType = PhoneCallLogID.getType();</pre>
Create a call log or conference call log.	<p>The PhoneCallLogID constructor removes dashes and other non-numeric characters from phone numbers.</p> <ol style="list-style-type: none"> 1. Create an instance of a PhoneCallLog or ConferencePhoneCallLog object, and provide the date, duration, participants, and notes for the call as parameters to the constructor. <pre>Date date = new Date("1000"); // date of call int duration = 60; // duration of call PhoneCallLogID caller1 = new PhoneCallLogID("555-1234"); // first participant PhoneCallLogID caller2 = new PhoneCallLogID("555-1235"); // second participant String notes = "New call."; // notes ConferencePhoneCallLog conferenceCall = new ConferencePhoneCallLog(date, duration, PhoneLogs.FOLDER_NORMAL_CALLS, caller1, caller2, notes);</pre> 2. Update the call log: <ul style="list-style-type: none"> • To update the call log, invoke PhoneLogs.addCall(CallLog call). <pre>_logs.addCall(conferenceCall);</pre> <ul style="list-style-type: none"> • To replace the call log with a new call log, invoke PhoneLogs.swapCall(CallLog call, int index, long folderID) <pre>_logs.swapCall(conferenceCall, 0, FOLDER_NORMAL_CALLS);</pre>
Delete a call log.	<p>> Invoke PhoneLogs.deleteCall().</p> <pre>_logs.deleteCall(0);</pre>

Code sample

Code sample: Calculating the time spent on the phone by a participant

Example: PhoneLogsDemo.java

```
/**
 * PhoneLogsDemo.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */
package com.rim.samples.docs.phonelogs;

import net.rim.blackberry.api.phone.phonelogs.*;
```

```

import java.lang.*;
import net.rim.device.api.system.Application;

public class PhoneLogsDemo extends Application
{
    private PhoneLogs _logs;
    private int _timeSpokenTo;

    static public void main(String[] args) {
        PhoneLogsDemo app = new PhoneLogsDemo();
        app.enterEventDispatcher();
    }

    private PhoneLogsDemo() {
        _logs = PhoneLogs.getInstance();
        PhoneCallLogID participant = new PhoneCallLogID("5551234");
        _timeSpokenTo = findTimeSpokenTo(participant,
            PhoneLogs.FOLDER_NORMAL_CALLS);
    }

    // Returns the number of seconds spent on the phone with a participant.
    public int findTimeSpokenTo(PhoneCallLogID participant,
        long folder) {
        int numberOfCalls = this._logs.numberOfCalls(folder);
        int timeSpokenTo = 0;
        PhoneCallLog phoneCallLog;
        ConferencePhoneCallLog conferencePhoneCallLog;
        for (int i = 0; i < numberOfCalls; i++) {
            Object o = _logs.callAt(i, folder);
            if (o instanceof PhoneCallLog) {
                phoneCallLog = (PhoneCallLog) o;
                if ( phoneCallLog.getParticipant() == participant)
                    timeSpokenTo += phoneCallLog.getDuration();
            } else {
                conferencePhoneCallLog = (ConferencePhoneCallLog) o;
                int participants = conferencePhoneCallLog.numberOfParticipants();
                for (int j = 0; j < participants; j++)
                    if (conferencePhoneCallLog.getParticipantAt(j) == participant) {
                        timeSpokenTo += conferencePhoneCallLog.getDuration();
                        j = participants;
                    }
            }
        }
        return timeSpokenTo;
    }
}

```

Using the BlackBerry Browser

Display content in the BlackBerry Browser
 Display content in a BlackBerry Browser field
 Code sample

Display content in the BlackBerry Browser

To display web content in the BlackBerry® Browser, use the `net.rim.blackberry.api.browser` package.

Task	Steps
Retrieve a BlackBerry® Browser session.	Retrieving the default session overrides any open sessions on the BlackBerry device. > Retrieve the default <code>BrowserSession</code> object by invoking the static method <code>Browser.getDefaultSession()</code> .
Retrieve a non-default BlackBerry® Browser session.	> Invoke <code>Browser.getSession()</code> .
Request a web page.	> Invoke <code>BrowserSession.displayPage(String url)</code> , specifying the URL that contains the web content. The following excerpt from the <code>Restaurants.java</code> sample creates a menu item that displays a web page in the BlackBerry Browser. <pre>private MenuItem browserItem = new MenuItem(_resources.getString(MENUITEM_BROWSER), 110, 12) { public void run() { synchronized(store) {String websiteUrl = websitefield.getText(); if (websiteUrl.length == 0) { Dialog.alert(_resources.getString(ALERT_NO_WEBSITE)); } else { BrowserSession visit = Browser.getDefaultSession(); visit.displayPage(websiteUrl); } } };</pre>

Display content in a BlackBerry Browser field

To display web content in a BlackBerry® Browser field, use the `net.rim.blackberry.api.browser` package.

Task	Steps
Access a rendering session.	1. Invoke <code>RenderingSession.getNewInstance()</code> . 2. Store the returned rendering session handle in a <code>RenderingSession</code> object. <pre>RenderingSession _renderingSession = RenderingSession.getNewInstance();</pre>

Task	Steps
Define callback functionality for a rendering session.	> Implement the <code>RenderingApplication</code> interface.
Retrieve a BlackBerry® Browser field.	<ol style="list-style-type: none"> 1. Invoke <code>RenderingSession.getBrowserContent(javax.microedition.io.HttpConnection, net.rim.device.api.browser.field.RenderingApplication, net.rim.device.api.browser.field.Event)</code>. 2. Store the returned object in a <code>BrowserContent</code> object. You render web content in the <code>BrowserContent</code> object.. <pre>BrowserContent browserContent = _renderingSession.getBrowserContent(HttpConnection connection, this, Event e);</pre>
Retrieve a field in which the URL content is rendered to your BlackBerry® Java® Application for display.	> Invoke <code>BrowserContent.getDisplayableContent()</code> , storing the returned object in a <code>Field</code> object. <pre>Field field = browserContent.getDisplayableContent();</pre>
Display a BlackBerry® Browser field.	<ol style="list-style-type: none"> 1. To clear the current screen, invoke the <code>MainScreen.deleteAll()</code> method. <pre>_mainScreen.deleteAll();</pre> 2. To add field data to the BlackBerry Java® Application screen, invoke <code>MainScreen.add()</code>. <pre>_mainScreen.add(field);</pre> 3. Create a non-main event thread to run <code>BrowserContent.finishLoading()</code> so that the UI does not lock. 4. To render the new BlackBerry browser content, invoke <code>BrowserContent.finishLoading()</code>. HTML files display a blank field until you invoke <code>BrowserContent.finishLoading()</code>. WML files and images might load before you invoke this method.
Create a separate thread for rendering.	> Create a non-main thread that contains the instructions for retrieving and displaying the BlackBerry® Browser field. <pre>class CreationThread extends Thread { BrowserFieldHandlerApplication _callBackApplication; BasicRenderingApplication _renderingApplication; public CreationThread(BrowserFieldHandlerApplication callBackApplication) { _callBackApplication = callBackApplication; } public void run() { _renderingApplication = new BasicRenderingApplication(_callBackApplication); BrowserField field = _renderingApplication.getBrowserField("www.blackberry.com"); _callBackApplication.displayBrowserField(field); } }</pre>
Set rendering options.	> Override <code>BrowserContent.getRenderingOptions()</code> . Your BlackBerry® Java® Application uses the default rendering options if you do not override <code>BrowserContent.getRenderingOptions()</code> .

Task	Steps
Manage events.	<p>> Implement of <code>RenderingApplication.eventOccurred()</code>, specifying the actions that occur when a specific rendering event occurs.</p> <p>The following example specifies actions that occur in the event of a URL request, change in browser content, or a redirect to a different web page.</p> <pre> public Object eventOccurred(Event event) { int eventId = event.getUID(); switch (eventId) { case Event.EVENT_URL_REQUESTED : { UrlRequestedEvent urlRequestedEvent = (UrlRequestedEvent) event; String absoluteUrl = urlRequestedEvent.getURL(); HttpConnection conn = null; PrimaryResourceFetchThread thread = new PrimaryResourceFetchThread(urlRequestedEvent.getURL(), urlRequestedEvent.getHeaders(), urlRequestedEvent.getPostData(), event, this); thread.start(); break;} case Event.EVENT_BROWSER_CONTENT_CHANGED: { // The browser field title might have changed, so we update the title field. BrowserContentChangedEvent browserContentChangedEvent = (BrowserContentChangedEvent) event; if (browserContentChangedEvent.getSource() instanceof BrowserContent) { BrowserContent browserField = (BrowserContent) browserContentChangedEvent.getSource(); String newTitle = browserField.getTitle(); if (newTitle != null) { _mainScreen.setTitle(newTitle);}} break; } case Event.EVENT_REDIRECT : { RedirectEvent e = (RedirectEvent) event; String referrer = e.getSourceURL(); switch (e.getType()) { case RedirectEvent.TYPE_JAVASCRIPT : break; case RedirectEvent.TYPE_META : // For MSIE and Mozilla, do not send a Referer for META Refresh. referrer = null; break; case Event.EVENT_SET_HEADER : // no cache support case Event.EVENT_SET_HTTP_COOKIE : // no cookie support default : } return null; } } } </pre>

Code sample

Code sample: Using the BlackBerry Browser

Example: BrowserFieldSampleApplication.java

```
/**
 * DefaultRenderingApplication.java
 * Copyright (C) 2004-2005 Research In Motion Limited.
 */

package com.rim.samples.docs.browser;

import java.io.IOException;
import javax.microedition.io.HttpConnection;
import net.rim.device.api.browser.field.*;
import net.rim.device.api.io.http.HttpHeaders;
import net.rim.device.api.system.Application;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.Status;
import net.rim.device.api.ui.container.MainScreen;

final public class BrowserFieldSampleApplication extends UiApplication implements
RenderingApplication
{
    private static final String REFERER = "referer";
    private RenderingSession _renderingSession;
    private MainScreen _mainScreen;
    private HttpConnection _currentConnection;

    public static void main(String[] args) {
        BrowserFieldSampleApplication app = new BrowserFieldSampleApplication();
        app.enterEventDispatcher();
    }

    private BrowserFieldSampleApplication() {
        _mainScreen = new MainScreen();
        pushScreen(_mainScreen);
        _renderingSession = RenderingSession.getNewInstance();

        PrimaryResourceFetchThread thread = new PrimaryResourceFetchThread("http://
www.google.com", null, null, null, this);
        thread.start();
    }

    public void processConnection(HttpConnection connection, Event e) {
        // cancel previous request
        if (_currentConnection != null) {
            try {
                _currentConnection.close();
            } catch (IOException e1) {
            }
        }
        _currentConnection = connection;
    }
}
```

```

        BrowserContent browserContent = null;

        try {
            browserContent = _renderingSession.getBrowserContent(connection, this, e);

            if (browserContent != null) {
                Field field = browserContent.getDisplayableContent();
                if (field != null) {
                    synchronized (Application.getEventLock()) {
                        _mainScreen.deleteAll();
                        _mainScreen.add(field);
                    }
                }
                browserContent.finishLoading();
            }
        } catch (RenderingException re) {
        } finally {
            SecondaryResourceFetchThread.doneAddingImages();
        }
    }

    /**
     * @see
     net.rim.device.api.browser.RenderingApplication#eventOccurred(net.rim.device.api.browser.E
     vent)
     */
    public Object eventOccurred(Event event) {
        int eventId = event.getUID();
        switch (eventId) {
            case Event.EVENT_URL_REQUESTED : {
                UrlRequestedEvent urlRequestedEvent = (UrlRequestedEvent) event;
                String absoluteUrl = urlRequestedEvent.getURL();

                HttpConnection conn = null;
                PrimaryResourceFetchThread thread = new
                PrimaryResourceFetchThread(urlRequestedEvent.getURL(),

                urlRequestedEvent.getHeaders(),

                urlRequestedEvent.getPostData(),

                event,

                this);

                thread.start();
                break;
            } case Event.EVENT_BROWSER_CONTENT_CHANGED: {
                // browser field title might have changed update title
                BrowserContentChangedEvent browserContentChangedEvent =
                (BrowserContentChangedEvent) event;
                if (browserContentChangedEvent.getSource() instanceof BrowserContent) {
                    BrowserContent browserField = (BrowserContent)
                browserContentChangedEvent.getSource();
                    String newTitle = browserField.getTitle();
                    if (newTitle != null) {
                        _mainScreen.setTitle(newTitle);
                    }
                }
                break;
            } case Event.EVENT_REDIRECT : {

```

```

        RedirectEvent e = (RedirectEvent) event;
        String referrer = e.getSourceURL();
        switch (e.getType()) {
            case RedirectEvent.TYPE_SINGLE_FRAME_REDIRECT :
                // show redirect message
                Application.getApplication().invokeAndWait(new Runnable() {
                    public void run() {
                        Status.show("You are being redirected to a different
page...");
                    }
                });
                break;
            case RedirectEvent.TYPE_JAVASCRIPT :
                break;
            case RedirectEvent.TYPE_META :
                // MSIE and Mozilla don't send a Referer for META Refresh.
                referrer = null;
                break;
            case RedirectEvent.TYPE_300_REDIRECT :
                // MSIE, Mozilla, and Opera all send the original
                // request's Referer as the Referer for the new
                // request.
                Object eventSource = e.getSource();
                if (eventSource instanceof HttpConnection) {
                    referrer =
((HttpConnection)eventSource).getRequestProperty(REFERER);
                }
                break;
        }
        HttpHeaders requestHeaders = new HttpHeaders();
        requestHeaders.setProperty(REFERER, referrer);
        PrimaryResourceFetchThread thread = new
PrimaryResourceFetchThread(e.getLocation(), requestHeaders, null, event, this);
        thread.start();
        break;
    } case Event.EVENT_CLOSE :
        // TODO: close the application
        break;
    case Event.EVENT_SET_HEADER :           // no cache support
    case Event.EVENT_SET_HTTP_COOKIE :      // no cookie support
    case Event.EVENT_HISTORY :               // no history support
    case Event.EVENT_EXECUTING_SCRIPT :      // no progress bar is supported
    case Event.EVENT_FULL_WINDOW :           // no full window support
    case Event.EVENT_STOP :                  // no stop loading support
    default :
    }

    return null;
}

/**
 * @see
net.rim.device.api.browser.RenderingApplication#getAvailableHeight(net.rim.device.api.brow
ser.BrowserContent)
 */
public int getAvailableHeight(BrowserContent browserField) {
    // field has full screen
    return Graphics.getScreenHeight();
}

```

```

    }

    /**
     * @see
     net.rim.device.api.browser.RenderingApplication#getAvailableWidth(net.rim.device.api.brows
     er.BrowserContent)
     */
    public int getAvailableWidth(BrowserContent browserField) {
        // field has full screen
        return Graphics.getScreenWidth();
    }

    /**
     * @see
     net.rim.device.api.browser.RenderingApplication#getHistoryPosition(net.rim.device.api.brow
     ser.BrowserContent)
     */
    public int getHistoryPosition(BrowserContent browserField) {
        // no history support
        return 0;
    }

    /**
     * @see
     net.rim.device.api.browser.RenderingApplication#getHTTPCookie(java.lang.String)
     */
    public String getHTTPCookie(String url) {
        // no cookie support
        return null;
    }

    /**
     * @see
     net.rim.device.api.browser.RenderingApplication#getResource(net.rim.device.api.browser.Req
     uestedResource,
     *     net.rim.device.api.browser.BrowserContent)
     */
    public HttpURLConnection getResource( RequestedResource resource, BrowserContent referrer)
    {

        if (resource == null) {
            return null;
        }

        // Verify that this is a cache-only request.
        if (resource.isCacheOnly()) {
            // no cache support
            return null;
        }

        String url = resource.getUrl();

        if (url == null) {
            return null;
        }

        // If the referrer is null, return the connection.
        if (referrer == null) {

```

```

        HttpURLConnection connection = Utilities.makeConnection(resource.getUrl(),
resource.getRequestHeaders(), null);
        return connection;
    } else {
        // If the referrer is not null, set up the connection on a separate thread.
        SecondaryResourceFetchThread.enqueue(resource, referrer);
    }
    return null;
}

/**
 * @see
net.rim.device.api.browser.RenderingApplication#invokeRunnable(java.lang.Runnable)
 */
public void invokeRunnable(Runnable runnable) {
    (new Thread(runnable)).run();
}

}

class PrimaryResourceFetchThread extends Thread
{
    private BrowserFieldSampleApplication _application;
    private Event _event;
    private byte[] _postData;
    private HttpHeaders _requestHeaders;
    private String _url;

    PrimaryResourceFetchThread(String url, HttpHeaders requestHeaders, byte[] postData,
                                Event event, BrowserFieldSampleApplication application) {

        _url = url;
        _requestHeaders = requestHeaders;
        _postData = postData;
        _application = application;
        _event = event;
    }

    public void run() {
        HttpURLConnection connection = Utilities.makeConnection(_url, _requestHeaders,
_postData);
        _application.processConnection(connection, _event);
    }
}

```

Creating push BlackBerry Java Applications

Types of push BlackBerry Java Applications
Types of push requests
Write a client push BlackBerry Java Application
Write a server-side push application
Create a RIM push request
Create a PAP push request
Code samples

Types of push BlackBerry Java Applications

Push applications send web content or data to specific BlackBerry® device users. Users do not need to request or download the data because the push application delivers the information as soon as it becomes available.

Two types of push applications exist:

Application	Description
Browser push applications	<p>Browser push applications send content to a web browser on the BlackBerry® device.</p> <ul style="list-style-type: none">• The BlackBerry Browser configuration supports BlackBerry MDS™ Services push applications.• The WAP Browser configuration supports WAP push applications.• The Internet Browser configuration does not support push applications. <p>See the <i>BlackBerry Browser Developer Guide</i> for more information about writing browser push applications.</p>
Client/server push applications	<p>A server-side application pushes data to a custom BlackBerry® Java® Application on the BlackBerry device. Client/server push applications consist of a custom client BlackBerry Java Application for the BlackBerry device and a server-side application that pushes content to the client BlackBerry Java Application. This approach provides more control than browser push applications over the type of content that you can send and how the BlackBerry device processes and displays the content.</p>

Types of push requests

Applications can send two types of push requests:

Request	Supported tasks	Push storage
RIM push	<ul style="list-style-type: none"> • sending a server-side push submission • specifying a reliability mode for the push submission • specifying a deliver-before time stamp for the push submission • requesting a result notification of the push submission • specifying a deliver-after time stamp for the push submission 	RIM pushes are stored in RAM. Undelivered RIM pushes might be lost if the server reboots.
PAP	<ul style="list-style-type: none"> • sending a server-side push submission • specifying a reliability mode for the push submission • specifying a deliver-before times tamp for the push submission • requesting a result notification of the push submission • specifying a deliver-after times tamp for the push submission • cancelling a push request submission • querying the status of a push request submission <p>Note: Part of the WAP 2.0 specification. For more information about PAP, visit http://www.openmobilealliance.org.</p>	PAP pushes are stored in a database.



Note: The BlackBerry® Mobile Data System™ queues only 1000 push requests, including both RIM and PAP push requests.

Write a client push BlackBerry Java Application

Task	Steps
Create a listening thread.	> Send and receive data on a separate thread so that you do not block the main event.
Determine if a BlackBerry® device is in a wireless coverage area.	<ol style="list-style-type: none"> Create code to check if the IPPP service book can be routed. <pre> if(ServiceBook.getSB().getRecordByUidAndCid(serviceUID, "IPPP") == null) { // There is no service book return false; } if(ServiceRouting.getInstance().isServiceRoutable(serviceUID, -1)) { // Serial bypass is active return true; } </pre> Create code to check that the wireless transceiver is on and that data coverage is on. <pre> return RadioInfo.getState() != RadioInfo.STATE_OFF && (RadioInfo.getNetworkService() & RadioInfo.NETWORK_SERVICE_DATA) != 0; </pre>
Open an input stream.	<ol style="list-style-type: none"> Invoke <code>Connector.open(String)</code>, specifying <code>http://</code> as the protocol and choosing a high port number from 1 to 65,535 to avoid conflicts with other applications. You cannot use the following port numbers: <ul style="list-style-type: none"> 80 443 7874 8080 <pre> Connector.open("http://6234"); </pre> To specify the connection type that the BlackBerry® Java® Application uses for incoming and outgoing connections, at the end of the connection string, add a colon, followed by the optional <code>deviceside=<i>boolean</i></code> parameter with one of the following values: <ul style="list-style-type: none"> If the BlackBerry Java Application listens for push information from BlackBerry® MDS™ Services, set the <code>deviceside=<i>boolean</i></code> parameter to <code>false</code>. If the BlackBerry Java Application listens for push information from WAP push requests, do not use the <code>deviceside=<i>boolean</i></code> parameter. Cast the object that <code>Connector.open</code> returns as a <code>StreamConnectionNotifier</code>. <pre> StreamConnectionNotifier _notify = (StreamConnectionNotifier)Connector.open("http://:6234"); </pre> Open a server-side stream connection once and keep the server-side stream connection open. <pre> // open a server-side stream connection StreamConnection stream = _notify.acceptAndOpen(); // open an input stream for the connection InputStream input = stream.openInputStream(); </pre> Read the incoming data. If you use application level push reliability, use the <code>pushInputStream.accept()</code> method to accept and acknowledge the incoming data. If an <code>IOException</code> occurs, reopen the connection.

Task	Steps
Close the stream connection notifier.	<pre>> Invoke <code>close()</code> on the stream connection notifier. <code>_notify.close();</code></pre>

See “Code sample: Listening for data from a web server” on page 211 for more information.

Write a server-side push application

You can use any programming language that can establish an HTTP connection to create a push application. The following sections use standard Java® to demonstrate a server-side push application.

Task	Steps
Specify a port.	<p>If you create an client/server push application, you must make sure that the server-side application uses a port number other than 80, 443, 7874, and 8080 to deliver push data.</p> <ul style="list-style-type: none"> > To specify a different port, in the application, include the X-Rim-Push-Dest-Port header with the port value.
Connect to the BlackBerry® MDS™ Connection Service.	<ul style="list-style-type: none"> > Establish a connection using the fully qualified computer name or IP address.
Construct the push URL.	<ul style="list-style-type: none"> > To create a push request, perform one of the following actions: <ul style="list-style-type: none"> • Create a RIM push request using the following format: <pre>/push?DESTINATION=<i>destination</i>&PORT=<i>port</i>&REQUESTURI=<i>uri</i> <headers> <content></pre> • Create a PAP push request using the following format: <pre>/pap</pre> <p>See “Create a RIM push request” on page 209 for more information about RIM push requests. See “Create a PAP push request” on page 210 for more information about PAP push requests.</p>
Connect to the BlackBerry® Enterprise Server.	<ol style="list-style-type: none"> 1. Invoke <code>openConnection()</code> on the push URL. 2. Cast the object that <code>url.openConnection()</code> returns as an <code>HttpURLConnection</code>. An <code>HttpURLConnection</code> represents a connection to a remote object. <pre>HttpURLConnection conn =(HttpURLConnection)url.openConnection();</pre>
Set properties for the HTTP POST request.	<ol style="list-style-type: none"> 1. Create a POST request. <pre>conn.setRequestMethod("POST"); // Post to the BlackBerry Enterprise Server.</pre> 2. To receive confirmation, set the parameter in <code>setDoInput(Boolean)</code> to <code>true</code> to indicate that the application intends to read data from the URL connection. <pre>conn.setDoInput(true);</pre> 3. To send data, set the parameter in <code>setDoOutput(Boolean)</code> to <code>true</code> to indicate that the application intends to send data to the URL connection. <pre>conn.setDoOutput(true);</pre>

Task	Steps
Write data to the server connection.	<ol style="list-style-type: none"> 1. To access an output stream, invoke <code>getOutputStream()</code>. <code>OutputStream out = conn.getOutputStream();</code> 2. Write data to the output stream. <code>out.write(data);</code> 3. Close the output stream. <code>out.close();</code>
Read the server response.	<ol style="list-style-type: none"> 1. To access an input stream, invoke <code>getInputStream()</code>. 2. Determine the size of the content. If the size of the content is non zero, open a data input stream, and then retrieve the content. <pre> InputStream ins = conn.getInputStream(); int contentLength = conn.getContentLength(); if (contentLength > 0) { byte[] someArray = new byte [contentLength]; DataInputStream dins = new DataInputStream(ins); dins.readFully(someArray); System.out.println(new String(someArray)); } ins.close(); </pre>
Close the server connection.	<p>> To indicate that the application will make no further requests to the server, invoke <code>disconnect()</code></p> <code>conn.disconnect();</code>

Work with a server-side push request

Send a request to cancel a PAP push submission.

1. Use the `cancel-message push-id` header. For example:
`<cancel-message push-id="123@wapforum.org">`
2. To specify the address to which the application submitted the push message, use the `address address-value` header. This is a required tag.
`<address address-value="WAPPUSH=aisha.wahl%40blackberry.com%3A7874/TYPE=USER@rim.net" />`

The following example shows a PAP push cancellation request:

```
Content-Type: application/xml
<?xml version="1.0"?>
<!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP 2.0//EN"
"http://www.wapforum.org/DTD/pap_2.0.dtd">
<pap>
<cancel-message push-id="a_push_id">
<address address-value=
"WAPPUSH=aisha.wahl%40blackberry.com%3A7874/TYPE=USER@rim.net"/>
</cancel-message>
</pap>
```

Note: When pushing to a Group address, you cannot determine the status of delivery to a particular recipient or cancel delivery to one or more recipients. If the application requires this functionality, specify multiple recipient addresses in the push submission.

Query the status of a PAP push request.

1. To specify the push message on which you want to obtain status information, send a PAP push query request using the `statusquery-message push-id` header. For example:
`<statusquery-message push-id="123@wapforum.org">`
2. To specify the address to which you want to submit the push message, in the PAP push query request, use the `address-value` header. For example:
`<address address-value="WAPPUSH=aisha.wahl%40blackberry.com%3A7874/TYPE=USER@rim.net" />`

The following example shows a RIM network status query request:

```
Content-Type: application/xml
<?xml version="1.0"?>
<!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP 2.0//EN"
"http://www.wapforum.org/DTD/pap_2.0.dtd"
[<?wap-pap-ver supported-versions="2.0,1.*"?>]>
<pap>
<statusquery-message push-id="123@wapforum.org">
<address
address-value="aisha.wahl%40blackberry.com%3A7874/TYPE=USER@rim.net"/>
</statusquery-message>
</pap>
```

Create a RIM push request

Task	Steps
Push content to one or multiple BlackBerry® device users using a RIM push request.	<ol style="list-style-type: none"> To push data to a single BlackBerry device user using RIM push, send an HTTP POST request using the following URL format. For example: <code>http://mdsServer:web_server_listen_port/push?DESTINATION=destination&PORT=port&REQUESTURI=uri headers content</code> where: <ul style="list-style-type: none"> <i>destination</i> is the destination PIN or email address <i>port</i> is the destination port number <i>uri</i> is the URI sent to the BlackBerry device <i>headers</i> consists of HTTP headers <i>content</i> is a byte stream To push content to multiple recipients using RIM push, include multiple DESTINATION parameters in the query string. <code>http://mds_server:8080/push?DESTINATION=user1@rim.com&DESTINATION=user2@rim.com&PORT=7874&REQUESTURI=</code>
Push content to a group using RIM push.	<p>> In the recipient addresses portion of the push submission, prefix the group name with the \$ character. In the following example, the \$ character is URL encoded.</p> <p>The following example shows a URL used to push to a group named IT using RIM push: <code>http://mds_server:8080/push?DESTINATION=%24IT&PORT=7874&REQUESTURI=</code></p>
Specify a unique message ID to cancel or check the status of a message.	<p>> Use the X-RIM-Push-ID header. Typically, specify a URL in combination with a value, such as 123@blackberry.com. If this header is omitted, the Mobile Data Service™ generates a unique message ID.</p> <p>Note: Push identifiers must not end in @ppg.rim.com.</p>
Specify a URL to send a result notification.	<p>> Use the X-RIM-Push-NotifyURL header.</p> <p>The result notification contains the X-RIM-Push-ID header, which specifies the message ID, and the X-RIM-Push-Status header, which specifies an HTTP response code. The notification also contains an X-RIM-Push-Destination header that specifies the recipient address to which the result pertains.</p>
Specify the delivery reliability mode of the content.	<p>> Use the X-RIM-Push-Reliability-Mode header with one of the following modes:</p> <ul style="list-style-type: none"> application-level (APPLICATION) application-preferred (APPLICATION-PREFERRED) transport-level (TRANSPORT)
Specify the date and time by which to deliver the content to the BlackBerry device.	<p>> Use the X-RIM-Push-Deliver-Before-timestamp header. Content that the application does not deliver before this date is not delivered. For example: <code>Mon, 28 Aug 2006 16:06:00 GMT</code></p>
Specify the date and time after which the content is delivered to the BlackBerry® device. The application does not deliver the content before this date. Represent the date and time in UTC format.	<p>> In the X-RIM-Push-Deliver-After-timestamp header, specify the date and time in UTC format. For example: <code>Mon, 28 Aug 2006 16:06:00 GMT</code></p>

Create a PAP push request

See "Appendix: XML control entity attributes" on page 265 for more information about XML control entity attributes.

Task	Steps
Push content to one or multiple BlackBerry® device users using PAP.	<ol style="list-style-type: none"> <p>To push data to a single BlackBerry device user using PAP, send an HTTP POST request using the following format:</p> <pre>http://mdsServer:web server listen port/pap</pre> <p>The URL to send the PAP push to.</p> <p>The request is a MIME multipart message, which consists of the following items:</p> <ul style="list-style-type: none"> XML document specifying the control entity push content <p>The following example shows a PAP push request:</p> <pre>Content-Type: multipart/related; type="application/xml"; boundary=asdlfkjiurwghasf X-Wap-Application-Id: / --asdlfkjiurwghasf Content-Type: application/xml <?xml version="1.0"?> <!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP 2.0//EN" "http:// www.wapforum.org/DTD/pap_2.0.dtd"> <pap> <push-message push-id="a_push_id" ppg-notify-requested-to="http:// foo.rim.net/ReceiveNotify"> <address address-value="WAPPUSH=aisha.wahl%40blackberry.com%3A7874/ TYPE=USER@rim.net"/> <quality-of-service delivery-method="unconfirmed"/> </push-message> </pap> --asdlfkjiurwghasf Content-Type: text/html <html><body>Hello, PAP world!</body></html> --asdlfkjiurwghasf--</pre> <p>To push content to multiple recipients using PAP, add multiple address tags to the post request. For example:</p> <pre><address address-value="WAPPUSH=user1%40rim%2ecom%5B%3A7874/ TYPE=USER@rim.net"></address> <address address-value="WAPPUSH=user2%40rim%2ecom%5D%3A7874/ TYPE=USER@rim.net"></address></pre>
Push content to a group using PAP.	<p>> In the recipient addresses part of the push submission, prefix the group name with the \$ character. In the following example, the \$ character is URL encoded.</p> <p>The following example shows an address element used to push to a group named IT using PAP:</p> <pre><address address-value="WAPPUSH=%24IT/TTYPE=USER@rim.net"/></pre>

See "Code sample: Pushing data to a BlackBerry Java Application that listens on a BlackBerry device" on page 214 for more information.

Code samples

Code sample: Listening for data from a web server

Example: HTTPPushDemo.java

```
/**
 * The client side of a simple HTTP Push system.
 * This application will listen for image data on the specified port and
 * render the data when it arrives.
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.httppush;

import java.io.*;
import javax.microedition.io.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.system.*;
import com.rim.samples.docs.resource.*;
import net.rim.device.api.util.*;
import net.rim.device.api.io.http.*;

public class HTTPPushDemo extends UiApplication implements HTTPPushDemoResResource
{
    // Constants.
    private static final String URL = "http://:100"; //PORT 100
    private static final int CHUNK_SIZE = 256;

    // Fields.
    private ListeningThread _listeningThread;
    private MainScreen _mainScreen;
    private RichTextField _infoField;
    private RichTextField _imageField;

    //statics -----
    private static ResourceBundle _resources =
ResourceBundle.getBundle(HTTPPushDemoResResource.BUNDLE_ID,
HTTPPushDemoResResource.BUNDLE_NAME);

    public static void main(String[] args) {
        HTTPPushDemo theApp = new HTTPPushDemo();
        theApp.enterEventDispatcher();
    }

    /**
     * Create a separate listening thread so that you do not
     * block the application's main event thread.
     */
    private class ListeningThread extends Thread {
        private boolean _stop = false;
        private StreamConnectionNotifier _notify;
```

```

    public synchronized void stop() {
        _stop = true;
        try {
            _notify.close(); // Close the connection so thread returns.
        } catch (IOException e) {
            System.err.println(e.toString());
        } catch (NullPointerException e) {
            // The notify object likely failed to open, due to an IOException.
        }
    }

    public void run()
    {
        StreamConnection stream = null;
        InputStream input = null;
        MDSPushInputStream pushInputStream=null;
        while (!_stop)
        {
            try
            {
                //synchronize here so that we don't end up creating a
                //connection that is never closed
                synchronized(this)
                {
                    // Open the connection once (or re-open after an IOException),
                    // so we don't end up in a race condition, where a push is lost if
                    // it comes in before the connection is open again.
                    // we open the url with a parameter that indicates that we should always
                    // use MDS when attempting to connect.
                    _notify = (StreamConnectionNotifier)Connector.open(URL + ";deviceside=false");
                }

                while (!_stop)
                {
                    //NOTE: the following will block until data is received
                    stream = _notify.acceptAndOpen();
                    try {
                        input = stream.openInputStream();
                        pushInputStream= new
MDSPushInputStream((HttpServerConnection)stream, input);

                        //Extract the data from the input stream

                        DataBuffer db = new DataBuffer();
                        byte[] data = new byte[CHUNK_SIZE];
                        int chunk = 0;
                        while ( -1 != (chunk = input.read(data)) )
                        {
                            db.write(data, 0, chunk);
                        }

                        updateMessage(data);
                        //This method is called to accept the push
                        pushInputStream.accept();
                    }
                }
            }
        }
    }

```

```

        input.close();
        stream.close();

        data = db.getArray();

    } catch (IOException e1) {
        // a problem occurred with the input stream
        // however, the original StreamConnectionNotifier is still valid
        System.err.println(e1.toString());

        if ( input != null ) {
            try {
                input.close();
            } catch (IOException e2) {
            }
        }
        if ( stream != null ) {
            try {
                stream.close();
            } catch (IOException e2) {
            }
        }
    }

    _notify.close();
    _notify = null;
} catch (IOException ioe)
{
    // likely the stream was closed
    System.err.println(ioe.toString());

    if ( _notify != null ) {
        try {
            _notify.close();
            _notify = null;
        } catch ( IOException e ) {
        }
    }
}

}

}

}

private final class HTTPMainScreen extends MainScreen
{
    public void close() {
        // Stop the listening thread.
        _listeningThread.stop();
        try {
            _listeningThread.join();
        } catch (InterruptedException e) {
            System.err.println(e.toString());
        }
        super.close();
    }
}

```

```

// Constructor.
public HTTPPushDemo() {
    _mainScreen = new HTTPMainScreen();
    _mainScreen.setTitle(new LabelField(_resources.getString(HTTPPUSHDEMO_TITLE),
LabelField.USE_ALL_WIDTH));
    _infoField = new RichTextField();
    _mainScreen.add(_infoField);
    _mainScreen.add(new SeparatorField());
    _imageField = new RichTextField();
    _mainScreen.add(_imageField);

    _listeningThread = new ListeningThread();
    _listeningThread.start();

    _infoField.setText(_resources.getString(HTTPPUSHDEMO_LISTENTHREADSTARTED));
    pushScreen(_mainScreen);
}

private void updateMessage(final byte[] data) {
    Application.getApplication().invokeLater(new Runnable() {
        public void run() {

            //query the user to load the received message
            String[] choices = {_resources.getString(HTTPPUSHDEMO_DIALOG_OK),
_resources.getString (HTTPPUSHDEMO_DIALOG_CANCEL)};
            if ( 0 != Dialog.ask(_resources.getString(HTTPPUSHDEMO_QUERYFORRENDER), choices, 0)
)
            {
                return;
            }
            _infoField.setText(_resources.getString(HTTPPUSHDEMO_IMAGEINFO) + data.length);

            try {
                _imageField.setText(new String(data));
            }
            catch (Exception e) {
                Dialog.inform(e.toString());
                System.err.println(e.toString());
            }
        }
    });
}
}

```

Code sample: Pushing data to a BlackBerry Java Application that listens on a BlackBerry device

The HTTPPush.java sample application, which uses standard Java®, sends a string of text to a listening client application on the BlackBerry® device using either a RIM push or a PAP push. The application pushes data based on an internet messaging address. To test push applications with the BlackBerry Device Simulator, define a mapping between the internet messaging address and the BlackBerry Device Simulator PIN (2100000A).

The following code sample compiles using J2SE 1.4.2.

Example: HTTPPushServer.java

```

/*
 * HttpPushServer.java
 * Copyright (C) 2001-2004 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.httppush;

import java.io.*;
import javax.swing.*;
import javax.swing.border.*;
import java.net.*;
import java.util.*;
import java.awt.Color;

/**
 * <p>The HTTPPushServer class provides a simple PUSH server sample.
 * <p>This program will send text to a listening device. The associated client demo
 * is HTTPPushServer. Start up both the device simulator and MDS before executing
 * this program. For reliable push, append the port that you are pushing to in
 * rimpublish.property file (push.application.reliable.ports):
 * <code>push.application.reliable.ports=7874,<b>100</b></code>
 *
 * <p> The general form of the URL for posting (pushing) data to the device is:
 * http://&lt;host&gt;:&lt;port&gt;/push?DESTINATION=&lt;device
 * pin&gt;&amp;PORT=&lt;device_port&gt;&REQUESTURI=&lt;post uri&gt;
 */
public class HTTPPushServer extends javax.swing.JFrame {

    //constants -----
    private static final String RESOURCE_PATH = "com/rim/samples/docs/httppush/resources";
    private static final String DEVICE_PIN = "2100000A";
    private static final String DEVICE_PORT = "100";
    private static final int MDS_PORT = 8080;

    private static final String IMAGE_TYPE = ".png";
    private String requestTemplate;
    private String notifyURL="http://localhost:7778";
    private Random random= new Random();
    private Thread notificationThread;

    //statics -----
    private static ResourceBundle _resources =
java.util.ResourceBundle.getBundle(RESOURCE_PATH);

    //constructors -----
    /** Creates a new HTTPPushServer instance*/
    public HTTPPushServer() {

        initComponents ();
        pack ();

        //sizing code for the main frame
        setSize(_panel.getWidth(), _panel.getHeight());

```

```

        setLocation(100,100);
        notificationThread= new NotificationThread();
    }

    private URL getPushURL(String DevicePin)
    {
        /**
         * The format of the URL is:
         * http://<host>:<port>/push?DESTINATION=<device
pin>&PORT=<device_port>&REQUESTURI=<post uri>
         */
        URL _pushURL = null;
        try {
            if ((DevicePin == null) || (DevicePin.length() == 0))
            {
                DevicePin = DEVICE_PIN;
            }
            _pushURL = new URL("http", "localhost", MDS_PORT, "/push?DESTINATION="+
DevicePin + "&PORT="+DEVICE_PORT+"&REQUESTURI=localhost");

            } catch (MalformedURLException e) {
                System.err.println(e.toString());
            }
            return _pushURL;
        }

        /**
         * This method is called from within the constructor to
         * initialize the form.
         * WARNING: Do NOT modify this code. The content of this method is
         * always regenerated by the FormEditor.
         */
        private void initComponents() { //GEN-BEGIN: initComponents
            _panel = new javax.swing.JPanel();
            _textField = new javax.swing.JTextField();
            _textArea= new javax.swing.JTextArea();
            _pinField = new javax.swing.JTextField(DEVICE_PIN);
            _label = new javax.swing.JTextArea();
            _notification=new javax.swing.JTextArea();
            _rimButton= new javax.swing.JRadioButton("rim");
            _papButton= new javax.swing.JRadioButton("pap");
            _buttonGroup= new javax.swing.ButtonGroup();
            _buttonGroup.add(_rimButton);
            _buttonGroup.add(_papButton);

            _sendButton = new javax.swing.JButton();
            getContentPane().setLayout(null);
            setTitle(java.util.ResourceBundle.getBundle("com/rim/samples/docs/httppush/
resources").getString("HTTPPushServer.title"));
            setResizable(false);
            addWindowListener(new java.awt.event.WindowAdapter() {
                public void windowClosing(java.awt.event.WindowEvent evt) {
                    exitForm(evt);
                }
            });

            _panel.setLayout(null);
            _panel.setPreferredSize(getSize());

```

```

        _textArea.setToolTipText(java.util.ResourceBundle.getBundle("com/rim/samples/docs/
httppush/resources").getString("HTTPPushServer._textField.toolTipText"));
        _panel.add(_textArea);
        _textArea.setBounds(10, 50, 270, 100);
        _textArea.setBorder(new LineBorder(Color.BLACK));

        _pinField.setToolTipText(java.util.ResourceBundle.getBundle("com/rim/samples/docs/
httppush/resources").getString("HTTPPushServer._pinField.toolTipText"));
        _panel.add(_pinField);
        _pinField.setBounds(10, 170, 150, 30);

        _panel.add(_rimButton);
        _panel.add(_papButton);
        _rimButton.setBounds(170, 170, 50, 30);
        _papButton.setBounds(240, 170, 50, 30);

        _label.setWrapStyleWord(true);
        _label.setLineWrap(true);
        _label.setEditable(false);
        _label.setText(java.util.ResourceBundle.getBundle("com/rim/samples/docs/httppush/
resources").getString("HTTPPushServer._label.text"));
        _label.setBackground((java.awt.Color) javax.swing.UIManager.getDefaults().get
("Button.background"));
        _panel.add(_label);
        _label.setBounds(10, 10, 270, 40);

        _sendButton.setLabel(java.util.ResourceBundle.getBundle("com/rim/samples/docs/
httppush/resources").getString("HTTPPushServer._sendButton.label"));
        _sendButton.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                sendButtonMouseClicked(evt);
            }
        });

        _panel.add(_sendButton);
        _sendButton.setLocation(10, 210);
        _sendButton.setSize(_sendButton.getPreferredSize());

        JScrollPane _scrollPane = new javax.swing.JScrollPane(_notification);
        _scrollPane.setVerticalScrollBarPolicy(
            JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

        _panel.add(_scrollPane);
        _scrollPane.setBounds(10, 250, 270, 150);

        getContentPane().add(_panel);
        _panel.setBounds(0, 0, 300, 450);
    } //GEN-END: initComponents

    private void sendButtonMouseClicked(java.awt.event.MouseEvent evt)
    { //GEN-FIRST:event_sendButtonMouseClicked

        String text = _textArea.getText();

```

```

        if(_rimButton.isSelected()) postData(text);
        else if(_papButton.isSelected()) papPush(text);

    } //GEN-LAST:event_sendButtonMouseClicked

    /**
     * <p>posts the specified data to the device
     * <p>The URL is hardcoded for the purposes of this demo, and takes the form:
     * http://<host>;<port>/push?DESTINATION=<device
     * pin>&PORT=<device_port>&REQUESTURI=<post uri>
     * param data the data to post
     */
    private void postData(String data)
    {

        String pushId="pushID:"+random.nextInt();

        setupNotifyThread();

        try {
            URL url = getPushURL(_pinField.getText());

            System.out.println(_resources.getString("HTTPPushServer.status.sendingToString") +
            url.toString());

            //open the connection using the static member...
            HttpURLConnection conn =(HttpURLConnection)url.openConnection();
            conn.setDoInput(true); //For receiving the confirmation
            conn.setDoOutput(true); //For sending the data
            conn.setRequestMethod("POST"); //Post the data to the proxy
            conn.setRequestProperty("X-RIM-PUSH-ID", pushId);
            conn.setRequestProperty("X-RIM-Push-NotifyURL", notifyURL);
            conn.setRequestProperty("X-RIM-Push-Reliability-Mode", "APPLICATION");
            //Write the data
            OutputStream out = conn.getOutputStream();
            out.write(data.getBytes());
            out.close();
            InputStream ins =conn.getInputStream();
            int contentLength =conn.getContentLength();
            System.out.println(
            _resources.getString("HTTPPushServer.status.contentLengthDescription")+ contentLength);
            if (contentLength > 0)
            {
                byte[] someArray = new byte [contentLength];
                DataInputStream dins = new DataInputStream(ins);
                dins.readFully(someArray);
                System.out.println(new String(someArray));
            }

            conn.disconnect();

        } catch (IOException e) {
            System.err.println(e);
        }
    }
}

```

```

private void readPapTemplate()
{
    try {
        String papFilename = "com/rim/samples/docs/httppush/pap_push.txt";
        InputStream ins = new BufferedInputStream(new FileInputStream(papFilename));
        ByteArrayOutputStream bouts = new ByteArrayOutputStream();
        copyStreams(ins, bouts);
        this.requestTemplate = new String(bouts.toByteArray());
    } catch (Exception exception) {
        exception.printStackTrace();
    }
}

private void setupNotifyThread()
{
    if( !notificationThread.isAlive() )
    {
        notificationThread = new NotificationThread();
        notificationThread.start();
    }
}

private void papPush(String data)
{
    String pushId="pushID:"+random.nextInt();

    setupNotifyThread();

    readPapTemplate();
    String errorCode = null;
    try {
        String mdsHost = "localhost";
        URL mdsUrl = new URL("http", mdsHost, MDS_PORT, "/pap");
        System.out.println(" sending PAP request to " + mdsUrl.toString() + "; pushId = "
+ pushId);

        HttpURLConnection mdsConn = (HttpURLConnection)mdsUrl.openConnection();

        String boundary = "";
        boundary = "asdlfkjiurwghasf";
        mdsConn.setRequestProperty("Content-Type", "multipart/related;
type=\"application/xml\"; boundary=" + boundary);
        mdsConn.setRequestProperty("X-Wap-Application-Id", "/");
        mdsConn.setRequestProperty("X-Rim-Push-Dest-Port","100");
        mdsConn.setRequestMethod("POST");

        mdsConn.setAllowUserInteraction(false);
        mdsConn.setDoInput(true);
        mdsConn.setDoOutput(true);

        String output = requestTemplate.replaceAll("\\$\\(pushid\\)", pushId);
        output = output.replaceAll("\\$\\(boundary\\)", boundary);
        output = output.replaceAll("\\$\\(notifyURL\\)", "" + notifyURL);
        output = output.replaceAll("\\$\\(pin\\)", "" + _pinField.getText());
    }
}

```

```

String    deliveryMethod = "confirmed";

output = output.replaceAll("\\$\\(deliveryMethod\\)", deliveryMethod);

output = output.replaceAll("\\$\\(headers\\)", "Content-Type: text/plain");
output = output.replaceAll("\\$\\(content\\)", data);

output = output.replaceAll("\\r\\n", "EOL");
output = output.replaceAll("\\n", "EOL");
output = output.replaceAll("EOL", "\\r\\n");

System.out.println(output);
OutputStream outs = mdsConn.getOutputStream();
copyStreams(new ByteArrayInputStream(output.getBytes()), outs);

mdsConn.connect();

ByteArrayOutputStream response = new ByteArrayOutputStream();
copyStreams(mdsConn.getInputStream(), response);

int httpCode = mdsConn.getResponseCode();

if (httpCode != HttpURLConnection.HTTP_ACCEPTED) {
    throw new Exception("MDS returned HTTP status: " + httpCode);
}

} catch (Exception exception) {
    if (errorCode == null)
    {
        errorCode = exception.getClass().getName();
    }

    System.out.println(" encountered error on submission: " +
exception.toString());
}

}

public void copyStreams(InputStream ins, OutputStream outs) throws IOException
{
    int maxRead = 1024;
    byte [] buffer = new byte[1024];
    int bytesRead;

    for(;;)
    {
        bytesRead = ins.read(buffer);
        System.out.println(buffer);
        if (bytesRead <= 0) break;
        outs.write(buffer, 0, bytesRead);
    }
}

/** Exit the Application */
private void exitForm(java.awt.event.WindowEvent evt) { //GEN-FIRST:event_exitForm
    System.exit (0);
} //GEN-LAST:event_exitForm

```

```

/**
 * @param args the command line arguments
 */
public static void main (String args[]) {
    new HTTPPushServer().show ();
}

// Variables declaration - do not modify//GEN-BEGIN:variables
private javax.swing.JPanel _panel;
private javax.swing.JTextField _textField;
private javax.swing.JTextArea _textArea;
private javax.swing.JTextField _pinField;
private javax.swing.JTextArea _label;
private javax.swing.JTextArea _notification;
private javax.swing.JButton _sendButton;
private javax.swing.JRadioButton _rimButton;
private javax.swing.JRadioButton _papButton;
private javax.swing.ButtonGroup _buttonGroup;
private javax.swing.JScrollPane _scrollPane;

// End of variables declaration//GEN-END:variables

public class NotificationThread extends Thread {

    public void run()
    {
        try {
            System.out.println("Waiting for notification on port " + 7778 + "...");
            while (true)
            {
                ServerSocket serverSocket = new ServerSocket(7778);
                serverSocket.setSoTimeout(120000);
                try {
                    Socket clientSocket = serverSocket.accept();
                    _notification.setText("Received notification:");
                    InputStream input = clientSocket.getInputStream();
                    StringBuffer str= new StringBuffer();
                    int byteRead = input.read();
                    while ((byteRead != -1) && (input.available() > 0))
                    {
                        str.append((char)byteRead);
                        byteRead = input.read();
                    }
                    _notification.append(str.toString());
                    PrintWriter output = new
PrintWriter(clientSocket.getOutputStream());
                    output.close();
                    clientSocket.close();
                } catch (SocketTimeoutException ste) {
                    System.out.println("Notification connection timeout.
Restarting...");
                }
                serverSocket.close();
            }
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }
}

```

```
    }  
}  
  

```

Localizing BlackBerry Java Applications

Storing text strings in resource files
Storing resources for a locale
Files required for localization
Add localization support
Retrieve strings from a resource file
Code samples

Storing text strings in resource files

Design BlackBerry® Java® Applications so that they are localized (adapt to specific languages and regions) without coding changes. Instead of including textual elements in your source code, store text strings in separate resource files. In your source code, use unique identifiers that map to the appropriate resource files.

Storing text strings in separate resource files has two benefits:

- Text translation is efficient because all of the text strings for a given locale are stored in a single file, outside your source code.
- BlackBerry Java Applications can dynamically retrieve the appropriate text to display to the BlackBerry device user, based on the locale of the BlackBerry device user.

The BlackBerry Java Development Environment includes a mechanism for creating string resources. The `net.rim.device.api.i18n` package includes the Localization API.



Note: MIDP applications do not support localization.

A `ResourceBundle` object stores the resources for a locale. A `ResourceBundleFamily` object contains a collection of `ResourceBundles`, which groups the resources for a BlackBerry Java Application. The BlackBerry Java Application can switch languages, depending on the locale of the BlackBerry device user, without requiring new resource bundles.

Storing resources for a locale

You can use the BlackBerry® Integrated Development Environment to compile each resource bundle into a separately compiled `.cod` file. You can load the appropriate `.cod` files onto BlackBerry devices with the `.cod` files for the BlackBerry Java® Application. The BlackBerry Integrated Development Environment organizes resources in a hierarchy based on inheritance. If you do not define a string in a locale, a string from the next closest locale is used.

Files required for localization

File required for localization	Description	Example
Resource header file	This file defines descriptive keys for each localized string. When the BlackBerry® Integrated Development Environment builds a project, it creates a resource interface with Resource appended to the .rrh file name. For example, if you create <code>AppName.rrh</code> , the interface is named <code>AppNameResource</code> .	<code>AppName.rrh</code>
Resource content file (root locale)	This file maps resource keys to string values for the root (global) locale. It has the same name as the resource header file.	<code>AppName.rrc</code>
Resource content file (specific locales)	This file maps resource keys to string values for specific locales (language and country). Files have the same name as the resource header file, followed by an underscore (_) and the language code, and then, optionally, an underscore (_) and country code. Two-letter language and country codes are specified in ISO-639 and ISO-3166, respectively.	<code>AppName_en.rrc</code> <code>AppName_en_GB.rrc</code> <code>AppName_fr.rrc</code>
Initialization file	This file initializes the resource bundle mechanism. This file is required only when you compile resources as a separate project.	<code>init.java</code>

Add localization support

Task	Steps
Add resource header files.	<ol style="list-style-type: none"> 1. In the BlackBerry IDE, select File -> New. 2. In the Files tab, select Other from the list of available file formats. 3. In the File name field, type a name. The file name should follow the proper format of the resource header file specified above, for example <i>ApplicationName.rrh</i>. 4. Browse to the location of the .java source files of the application and then click OK. The path should appear in the Location field. 5. Click OK. 6. In the Add resource header (.rrh) file, ensure that the package name has been populated correctly. 7. Click OK. 8. In the text editor, confirm that the resource header file displays the package name correctly. 9. Close the file. 10. Right-click the project and select Add File to Project Name, where <i>Project Name</i> is the name of the project. 11. From the Files of Type drop-down list, browse to the location of the resource header file and select *.rrh. 12. Highlight the resource header file and select Open.

Task	Steps
Add resource content files.	<p>Create resource content files in the folder where the .java file exists. For example, in the folder that contains CountryInfo.java, create CountryInfo.rrc (root locale), CountryInfo_en.rrc (English), and CountryInfo_fr.rrc (French).</p> <ol style="list-style-type: none"> 1. In the BlackBerry IDE, select File -> New. 2. In the Files tab, select Other from the list of available file formats. 3. In the File name field, type a name of the root resource content file. This name should correspond to the application name with the extension .rrc (i.e. <i>ApplicationName.rrc</i>). For example, <i>_en.rrc(English)</i> or <i>_fr.rrc(french)</i>. 4. Browse to the location of the .java source files of the application and then click OK. The path should appear in the Location field. 5. Click OK. 6. If prompted to save changes, select Yes. 7. Right-click the project and select Add File to Project Name, where <i>Project Name</i> is the name of the project. 8. From the Files of Type drop-down list, browse to the location of the resource content file and select *.rrc. 9. Highlight the resource content file and select Open.
Add resources.	<ol style="list-style-type: none"> 1. In the BlackBerry® IDE, double-click a resource header file. 2. Add resource keys: <ul style="list-style-type: none"> • To add one value per key: On the Root tab, type resource keys and values for each string or string array in your BlackBerry® MDS Java® Application. • To add multiple values per key: In the resource editor, right-click a resource and click Convert to Multiple Values. Add one or more values to the array. 3. To specify a different text string in other locales, select the tab for a locale, such as fr for the French language. 4. In the Value cell for the resource, type the text string for the locale. If you do not define a value for a resource in a particular locale, the value for the root locale is used. You can type unicode characters directly into the Value cell. For more information about unicode characters, visit http://www.unicode.org.
Display a localized BlackBerry® Java® Application title on the Home screen.	<p>If you do not provide a resource for the BlackBerry Java Application title, the BlackBerry Java Application uses the value you type in the Title field on the Application tab of the Project Properties window as the BlackBerry Java Application title.</p> <ol style="list-style-type: none"> 1. In the resource editor, add a resource for the BlackBerry Java Application title, such as APPLICATION_TITLE. 2. Type a value for this resource in each locale that you support. 3. In the BlackBerry® IDE, right-click the BlackBerry Java Application project, and then click Properties. 4. On the Resources tab, select the Title Resource Available option. 5. From the Resource Bundle drop-down list, select the resource header file name to use for this BlackBerry Java Application. 6. From the Title Id drop-down list, select the resource to use for the BlackBerry Java Application title, such as APPLICATION_TITLE. 7. From the Description Id drop-down list, select a description ID.

See "Code sample: Storing text strings in separate resources for locales" on page 228 for more information.

Retrieve strings from a resource file

Task	Steps
Retrieve the resource bundle.	<p>When the BlackBerry® Integrated Development Environment builds your project, it creates an interface for each .rrh resource file.</p> <ol style="list-style-type: none"> 1. Import the interfaces that the BlackBerry IDE creates. <pre>import com.rim.samples.docs.resource.CountryInfoResource;</pre> 2. Create a <code>ResourceBundle</code> object to contain the localized resources, such as strings, for a BlackBerry Java® Application. <pre>private static ResourceBundle _resources = ResourceBundle.getBundle(CountryInfoResource.BUNDLE_ID, CountryInfoResource.BUNDLE_NAME);</pre> 3. To retrieve the appropriate bundle family, invoke <code>getBundle()</code>.
Create menu items using resources.	<p>> Use the <code>MenuItem</code> constructor that accepts a resource bundle and a resource instead of a <code>String</code> for the name of the menu item.</p> <pre>private MenuItem _viewItem = new MenuItem(_resources, MENUITEM_VIEW, 110, 10) { public void run() { select = choiceField.getSelectedIndex(); _infoScreen = new InfoScreen(); UiApplication.getUiApplication().pushScreen(_infoScreen); } }</pre>
Replace text strings with the appropriate resources.	<ol style="list-style-type: none"> 1. For each field that appears on the main screen, replace the text string with the appropriate resource. 2. Invoke <code>getString()</code> or <code>getStringArray()</code> to retrieve the string for the appropriate language. <pre>LabelField title = new LabelField(_resources.getString(APPLICATION_TITLE), LabelField.ELLIPSIS LabelField.USE_ALL_WIDTH); add(new RichTextField(_resources.getString(FIELD_TITLE))); String choices[] = _resources.getStringArray(FIELD_COUNTRIES); choiceField = new ObjectChoiceField(_resources.getString(FIELD_CHOICE), choices);</pre>

See "Code sample: Retrieving strings from a resource file" on page 230 for more information.

Manage resource files for BlackBerry Java Application suites

If you are creating a suite of BlackBerry® Java® Applications, organize resources into separate projects for each locale.

Task	Steps
Create resource projects.	<ol style="list-style-type: none"> 1. Open the BlackBerry® Integrated Development Environment. 2. Create a project for each resource bundle (locale), including the root locale. 3. Give the projects for each locale the same name as the project for the root locale, followed by a double underscore (__), the language code, and, optionally, an underscore (_) followed by the country code. For example, if the root locale project is named <code>com_company_app</code>, the projects for each locale would be named <code>com_company_app__en</code>, <code>com_company_app__en_GB</code>, and <code>com_company_app__fr</code>.
Specify output file names.	<p>The output file names for all resource locale projects must be the same as for the root locale, followed by a double underscore and the appropriate language and country codes.</p> <ol style="list-style-type: none"> 1. Right-click the project, and then click Properties. 2. On the Build tab, in the Output file name field, type a name for the compiled file, without a file name extension.
Create an initialization file.	<p>> The BlackBerry® IDE provides a built-in initialization mechanism, so that you only need to create an empty initialization class with an empty <code>main()</code>.</p> <pre>package com.rim.samples.device.resource; import net.rim.device.api.i18n.*; public class init { public static void main (String[] args) { }</pre>
Add files to appropriate resource projects.	<p>If you support a large number of locales, create a single library project for all resource header (.rrh) files and set the project type to Library. For each resource locale in this project, define a dependency between the projects.</p> <ol style="list-style-type: none"> 1. Create one resource header file for each BlackBerry® Java® Application. 2. Add the resource header (.rrh) files to the project for each BlackBerry Java Application. 3. Add the resource header files to each resource project. 4. Create one resource content file for each BlackBerry Java Application. 5. Create one resource content file for each supported locale. 6. In each resource project, right-click each .rrh file, and then click Properties. 7. Select Dependency only. Do not build. 8. Add the resource content (.rrc) files to the projects for the appropriate locales.

Code samples

Code sample: Storing text strings in separate resources for locales

Example: CountryInfo.java

```
/**
 * CountryInfo.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */
package com.rim.samples.docs.countryinfo;

import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import net.rim.device.api.i18n.*;
import com.rim.samples.docs.resource.*;

/* This sample demonstrates how to store text strings in separate resource
   files for specific locales rather than providing text strings directly
   in the code. In your source code, you retrieve the string from the resource
   to display the appropriate text for the user locale. */
public class CountryInfo extends UiApplication {
    public static void main(String[] args) {
        CountryInfo theApp = new CountryInfo();
        theApp.enterEventDispatcher();
    }

    public CountryInfo() {
        pushScreen(new HelloWorldScreen());
    }
}

final class HelloWorldScreen extends MainScreen implements CountryInfoResource {
    private InfoScreen _infoScreen;
    private ObjectChoiceField choiceField;
    private int select;

    private static ResourceBundle _resources = ResourceBundle.getBundle(
        CountryInfoResource.BUNDLE_ID, CountryInfoResource.BUNDLE_NAME);

    public HelloWorldScreen() {
        super();
        LabelField title = new LabelField(_resources.getString(APPLICATION_TITLE),
            LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);
        setTitle(title);
        add(new RichTextField(_resources.getString(FIELD_TITLE)));
        String choices[] = _resources.getStringArray(FIELD_COUNTRIES);
        choiceField = new ObjectChoiceField(
            _resources.getString(FIELD_CHOICE), choices);
        add(choiceField);
    }

    public boolean onClose() {
```

```

        Dialog.alert(_resources.getString(CLOSE));
        System.exit(0);
        return true;
    }

    private MenuItem _viewItem = new MenuItem(_resources, MENUITEM_VIEW, 110, 10) {
        public void run() {
            select = choiceField.getSelectedIndex();
            _infoScreen = new InfoScreen();
            UiApplication.getUiApplication().pushScreen(_infoScreen);
        }
    };

    private MenuItem _closeItem = new MenuItem(_resources, MENUITEM_CLOSE,
        200000, 10) {
        public void run() {
            onClose();
        }
    };

    protected void makeMenu( Menu menu, int instance ) {
        menu.add(_viewItem);
        menu.add(_closeItem);
    }

    private class InfoScreen extends MainScreen {
        public InfoScreen() {
            super();
            LabelField lf = new LabelField();
            BasicEditField popField = new BasicEditField(
                _resources.getString(FIELD_POP), null, 20, Field.READONLY);
            BasicEditField langField = new BasicEditField(
                _resources.getString(FIELD_LANG), null, 20, Field.READONLY);
            BasicEditField citiesField = new BasicEditField(
                _resources.getString(FIELD_CITIES), null, 50, Field.READONLY);
            add(lf);
            add(new SeparatorField());
            add(popField);
            add(langField);
            add(citiesField);
            if (select == 0) {
                lf.setText(_resources.getString(FIELD_US));
                popField.setText(_resources.getString(FIELD_US_POP));
                langField.setText(_resources.getString(FIELD_US_LANG));
                citiesField.setText(_resources.getString(FIELD_US_CITIES));
            } else if (select == 1) {
                lf.setText(_resources.getString(FIELD_CHINA));
                popField.setText(_resources.getString(FIELD_CHINA_POP));
                langField.setText(_resources.getString(FIELD_CHINA_LANG));
                citiesField.setText(_resources.getString(FIELD_CHINA_CITIES));
            } else if (select == 2) {
                lf.setText(_resources.getString(FIELD_GERMANY));
                popField.setText(_resources.getString(FIELD_GERMANY_POP));
                langField.setText(_resources.getString(FIELD_GERMANY_LANG));
                citiesField.setText(
                    _resources.getString(FIELD_GERMANY_CITIES));
            }
        }
    }
}

```

```

    }
}

```

Code sample: Retrieving strings from a resource file

Example: CountryInfo.java (with localization support)

```

/**
 * CountryInfo.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */
package com.rim.samples.docs.countryinfo;

import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import net.rim.device.api.i18n.*;
import com.rim.samples.docs.resource.*;

/* This sample demonstrates how to store text strings in separate resource
   files for specific locales rather than providing text strings directly
   in the code. In your source code, you retrieve the string from the resource
   to display the appropriate text for the user locale. */
public class CountryInfo extends UiApplication {
    public static void main(String[] args) {
        CountryInfo theApp = new CountryInfo();
        theApp.enterEventDispatcher();
    }

    public CountryInfo() {
        pushScreen(new HelloWorldScreen());
    }
}

final class HelloWorldScreen extends MainScreen implements CountryInfoResource {
    private InfoScreen _infoScreen;
    private ObjectChoiceField choiceField;
    private int select;

    private static ResourceBundle _resources = ResourceBundle.getBundle(
        CountryInfoResource.BUNDLE_ID, CountryInfoResource.BUNDLE_NAME);

    public HelloWorldScreen() {
        super();
        LabelField title = new LabelField(_resources.getString(APPLICATION_TITLE),
            LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);
        setTitle(title);
        add(new RichTextField(_resources.getString(FIELD_TITLE)));
        String choices[] = _resources.getStringArray(FIELD_COUNTRIES);
        choiceField = new ObjectChoiceField(
            _resources.getString(FIELD_CHOICE), choices);
        add(choiceField);
    }
}

```

```

}

public boolean onClose() {
    Dialog.alert(_resources.getString(CLOSE));
    System.exit(0);
    return true;
}

private MenuItem _viewItem = new MenuItem(_resources, MENUITEM_VIEW, 110, 10) {
    public void run() {
        select = choiceField.getSelectedIndex();
        _infoScreen = new InfoScreen();
        UiApplication.getUiApplication().pushScreen(_infoScreen);
    }
};

private MenuItem _closeItem = new MenuItem(_resources, MENUITEM_CLOSE,
    200000, 10) {
    public void run() {
        onClose();
    }
};

protected void makeMenu( Menu menu, int instance ) {
    menu.add(_viewItem);
    menu.add(_closeItem);
}

private class InfoScreen extends MainScreen {
    public InfoScreen() {
        super();
        LabelField lf = new LabelField();
        BasicEditField popField = new BasicEditField(
            _resources.getString(FIELD_POP), null, 20, Field.READONLY);
        BasicEditField langField = new BasicEditField(
            _resources.getString(FIELD_LANG), null, 20, Field.READONLY);
        BasicEditField citiesField = new BasicEditField(
            _resources.getString(FIELD_CITIES), null, 50, Field.READONLY);
        add(lf);
        add(new SeparatorField());
        add(popField);
        add(langField);
        add(citiesField);
        if (select == 0) {
            lf.setText(_resources.getString(FIELD_US));
            popField.setText(_resources.getString(FIELD_US_POP));
            langField.setText(_resources.getString(FIELD_US_LANG));
            citiesField.setText(_resources.getString(FIELD_US_CITIES));
        } else if (select == 1) {
            lf.setText(_resources.getString(FIELD_CHINA));
            popField.setText(_resources.getString(FIELD_CHINA_POP));
            langField.setText(_resources.getString(FIELD_CHINA_LANG));
            citiesField.setText(_resources.getString(FIELD_CHINA_CITIES));
        } else if (select == 2) {
            lf.setText(_resources.getString(FIELD_GERMANY));
            popField.setText(_resources.getString(FIELD_GERMANY_POP));
            langField.setText(_resources.getString(FIELD_GERMANY_LANG));
            citiesField.setText(

```

```
        _resources.getString(FIELD_GERMANY_CITIES));  
    }  
}  
}
```

Testing BlackBerry Java Applications

Testing BlackBerry Java Applications using the BlackBerry IDE
Testing BlackBerry Java Applications using BlackBerry devices
Debugging BlackBerry Java Applications

Testing BlackBerry Java Applications using the BlackBerry IDE

After you develop and compile your BlackBerry® Java® Application, you should test it on the BlackBerry device. The most common first step is to set up the BlackBerry Integrated Development Environment to use a BlackBerry Device Simulator for testing. The BlackBerry Device Simulators run the same Java code as the live BlackBerry devices, so they provide an environment for testing how BlackBerry Java Applications will function on a live BlackBerry device. Each version of the BlackBerry Java Development Environment comes with the BlackBerry Device Simulators that are available when Research In Motion released that version of the BlackBerry JDE. You can download additional BlackBerry Device Simulators as they are available on the BlackBerry Developer Zone.

Use the BlackBerry Device Simulator to test synchronizing data with the BlackBerry Desktop Software

To complete the following instructions, the BlackBerry® Desktop Manager must exist on your computer. See the *BlackBerry Integrated Development Environment Online Help* for information on starting the BlackBerry Device Simulator when you run a BlackBerry Java® Application in the BlackBerry Integrated Development Environment.

Task	Steps
Set the connection.	<ol style="list-style-type: none">1. In the Desktop Manager window, on the Options menu, click Connection Settings.2. In the Connection drop-down list, click USB.3. Click OK.4. Close the BlackBerry® Desktop Manager.
Run the BlackBerry® Java® Application.	<ol style="list-style-type: none">1. Open the BlackBerry Integrated Development Environment.2. From the main menu, click Preferences.3. In the Ports tab, select USB cable connected.4. Click OK.5. Build and run the BlackBerry Java Application in the BlackBerry IDE.
Detect the BlackBerry Device Simulator.	<ol style="list-style-type: none">1. After the BlackBerry® Device Simulator starts, start the BlackBerry Desktop Manager.2. In the BlackBerry Desktop Manager window, on the Options menu, click Connection Settings.3. Click Detect to detect the BlackBerry Device Simulator.

Testing BlackBerry Java Applications using BlackBerry devices

After testing your BlackBerry® Java® Application on the BlackBerry Device Simulator, load your BlackBerry Java Application on a live BlackBerry device. When the BlackBerry Java Application loads, you can open and test its functionality and performance. For debugging purposes, attach your BlackBerry device to the BlackBerry Integrated Development Environment debugger to step through your BlackBerry Java Application code. This can be useful if you are trying to identify a network issue, a Bluetooth wireless technology issue, or other items that are difficult to simulate accurately.

If your BlackBerry Java Application uses signed APIs, you might require code signing keys. See the *BlackBerry Java Development Environment Fundamentals Guide* for more information about code signing keys.

Connect the BlackBerry IDE to a BlackBerry device

1. Connect a BlackBerry® device to the computer.
2. Open the BlackBerry Integrated Development Environment.
3. To connect a BlackBerry device that uses a USB port, click **Debug > Attach to > Handheld > USB<PIN>**, where *PIN* is the PIN of the BlackBerry device that is connected to the computer.

Debugging BlackBerry Java Applications

When you connect a BlackBerry® device to a computer to perform testing and optimization of code, run BlackBerry Java® Applications on the BlackBerry device and use the BlackBerry Integrated Development Environment debug tools.

Use breakpoints

In the breakpoints pane, you can perform any of the following actions:

Task	Steps
Execute code and print values to the Output window when code execution reaches a Breakpoint.	> In the Execute when hit field, type an expression. For example, type <code>System.out.println(foo)</code> .
Open the source code at a set breakpoint.	> In the Resume if true field, type an expression. When BlackBerry® Java® Application execution reaches the breakpoint, BlackBerry Java Application execution resumes if the expression evaluates to true.
Stop the BlackBerry® Java® Application after a specific number of iterations through a breakpoint.	> In the Iteration field, type a positive integer. When you start debugging, execution stops when the number of iterations through a breakpoint equals the number you typed.

Task	Steps
Stop the BlackBerry® Java® Application when a condition is true.	> In the Condition field, type a Boolean expression, such as <code>x==100</code> . The Hits field calculates the number of times the BlackBerry Java Application stops at a breakpoint when the Condition is true or the Condition field is empty.
Remove a breakpoint.	<ol style="list-style-type: none"> 1. Open the source file. 2. In the Edit window, click the line of code that contains the breakpoint to remove. 3. On the Debug menu, click Breakpoint > Delete Breakpoint at Cursor.
Remove all breakpoints.	<ol style="list-style-type: none"> 1. On the View menu, click Breakpoints. 2. In the breakpoints pane, click Delete All.

One debugging method is to start by setting only a few breakpoints at critical sections of your code, and then gradually set breakpoints at shorter intervals to identify the problem. To identify the problem, after the BlackBerry® Java® Application pauses at a breakpoint, use debugging tools to view various BlackBerry Java Application processes and statistics.

Debug a BlackBerry Java Application in the BlackBerry IDE

1. Copy the BlackBerry Java Application .cod, .csf, .cso, .debug, and .jar files into the BlackBerry® Device Simulator root directory:

`C:\Program Files\Research In Motion\BlackBerry JDE 4.3\simulator`

2. Open the BlackBerry Integrated Development Environment.
3. Add and set up a workspace for your BlackBerry Java Application.
4. Add your BlackBerry Java Application to the workspace.
5. Start any simulators that your BlackBerry Java Application requires.

Without access to a BlackBerry Enterprise Server, you need the BlackBerry MDS™ Simulator to simulate browser traffic, HTTP/TCP connections to third-party applications, and push functionality. To debug BlackBerry Java Applications that send and receive messages between a BlackBerry Device Simulator and a computer email application, use the BlackBerry email server simulator. See the *BlackBerry Device Simulator User Guide* for more information.

6. On the **Debug** menu, click **Go**. The BlackBerry IDE builds all active projects in the workspace and loads the BlackBerry Java Applications in the BlackBerry Device Simulator.
7. In the BlackBerry Device Simulator window, test the relevant sections of code.
8. On the **Debug** menu, click **Break Now**.
9. To resume debugging your BlackBerry Java Application, on the **Debug** menu, click **Continue**.
10. To stop debugging your BlackBerry Java Application, in the BlackBerry Device Simulator, on the **File** menu, click **Quit**.

Manage a debugging session

Task	Steps
Continue a debugging session.	> To resume running the BlackBerry® Java® Application, on the Debug menu, click Continue .
End a debugging session in the BlackBerry® Device Simulator.	> In the BlackBerry Device Simulator, on the File menu, click Quit .
End a debugging session in the BlackBerry® Integrated Development Environment.	<ol style="list-style-type: none"> 1. In the main window, on the Debug menu, click Stop Debugging. 2. Read the warning message. 3. If you agree, click Yes.
Interrupt a debugging session without stopping a BlackBerry® Java® Application.	> In the main window, on the Debug menu, click Break Now .

Locate an error in the source code

1. In the Output window, double-click the error message.
2. Perform one of the following actions:

Task	Steps
View the next error.	> Press F4 .
Return to the previous error.	> Press SHIFT+F4

Run a BlackBerry Java Application to the insertion point

1. In the Edit window, click the line of code at which you want to stop the BlackBerry® Java® Application.
2. In the **Debug** menu, click **Run to Cursor**.



Tip: To stop execution at a specific location, set a breakpoint on a line of code.

Debug a BlackBerry Java Application on a BlackBerry device

To perform testing and optimization for BlackBerry® Java® Applications on a BlackBerry device that is connected to your computer, use the BlackBerry Integrated Development Environment debugging tools.



Note: To connect the BlackBerry Integrated Development Environment to a BlackBerry device using a USB port, install the BlackBerry Desktop Software Version 3.5.1 or later.

Install .debug files on your computer

To debug BlackBerry® Java® Applications using a BlackBerry device, the .debug files in the BlackBerry Integrated Development Environment must match the software version number of the BlackBerry device. BlackBerry Device Simulator packages, which you can download from the BlackBerry Developer Zone, contain .debug files for specific BlackBerry devices.

1. Download the BlackBerry Device Simulator package for your BlackBerry device software version number from the BlackBerry Developer Zone at:
<http://blackberry.com/developers/downloads/simulators>
2. Connect a BlackBerry device to the computer. See “Connect the BlackBerry Integrated Development Environment debugger to a BlackBerry device” on page 239 for more information about connecting a BlackBerry device to a computer.
3. In the BlackBerry IDE, on the **Edit** menu, click **Preferences**.
4. Click the **Debug** tab.
5. Click the **Other** tab.
6. In the **Handheld debug file location** field, type the location of the downloaded .debug files. The .debug files are located in the Debug directory of the BlackBerry Device Simulator package installation directory.

Load BlackBerry Java Applications onto a BlackBerry device

For development and testing purposes, use JavaLoader.exe to load BlackBerry® Java® Applications onto a BlackBerry device.

1. Exit the BlackBerry Desktop Software.
2. Connect the BlackBerry device to the computer.
3. At a command prompt, switch to the following directory:
c:\Program Files\Research In Motion\BlackBerry JDE 4.3.0\bin
4. Type the following command:
JavaLoader [-usb] [-pport] [-bbps] [-wpassword] load file

Parameter	Description
<i>port</i>	A BlackBerry device PIN if the BlackBerry device connects to a USB port. You must also specify the -usb option.
<i>password</i>	This parameter specifies the password for the BlackBerry device, if a password for the BlackBerry device is set.
<i>file</i>	This parameter specifies one or more .cod files to load onto the BlackBerry device.



Note: BlackBerry device users should use the BlackBerry Desktop Software to load BlackBerry Java Applications onto their BlackBerry devices.

Connect the BlackBerry IDE to a BlackBerry device

- > In the BlackBerry® IDE, on the **Debug** menu, click **Attach to > Handheld > USB (PIN)**, where *PIN* is the PIN of the BlackBerry device.

You can now run your BlackBerry Java® Applications on the BlackBerry device and use the BlackBerry Integrated Development Environment debugging tools to test and optimize your BlackBerry Java Application.

Step through lines of code in a BlackBerry Java Application

In the main window, on the **Debug** menu, perform any of the following commands:

Task	Steps
Step over a method call.	The BlackBerry® Integrated Development Environment debugger moves to the next line of code. If the source line is a method call, the BlackBerry Java® Application runs the entire method without stepping through the individual method instructions.
Step through method instructions.	The BlackBerry® IDE debugger moves to the next line of code. If the source line is a method call, the BlackBerry Java® Application stops just before running the first statement of the method.
Step out of method instructions.	The BlackBerry® IDE debugger moves to the next line of code. If the source line is part of a method, the BlackBerry Java® Application runs the remaining lines of the method and returns control to the caller of the method.

For example, to step into function "f" in the following line of code `f(g(x))`, perform the following actions:

1. Click **Step Into** to run the BlackBerry Java Application into "g"
2. Click **Step Out** to return the BlackBerry Java Application to the line of code.
3. Click **Step Into** again to run the BlackBerry Java Application into function "f"

View statistics to locate memory leaks

To locate memory leaks, use the Memory Statistics tool with the Objects tool. Begin by using the Memory Statistics tool to retrieve information on the memory usage of your BlackBerry® Java® Application. The Memory Statistics tool identifies the number of objects in memory, while the Objects tool displays detailed information for each object.

The Memory Statistics tool displays the statistics on the number of objects and bytes in use for object handles, RAM, and flash memory.

Locate a memory leak

1. Set two or more breakpoints in your code.
2. Open the BlackBerry® Integrated Development Environment.
3. On the **Debug** menu, click **Go**. The BlackBerry Java® Application runs to the first breakpoint.
4. In the main window, on the **View** menu, click **Memory Statistics**.
5. In the memory statistics pane, click **Refresh**.
6. Click **Snapshot**.
7. On the **Debug** menu, click **Continue**. The BlackBerry Java Application runs to the second breakpoint.
8. In the memory statistics pane, click the **Refresh** tab.
9. Click **Compare to Snapshot**.
10. Repeat steps 1 through 8, setting breakpoints closer together until they converge on the memory leak.

Display objects in memory to locate object leaks

Object leaks can cause the BlackBerry® JVM to run out of flash memory, which forces a BlackBerry device to reset.

Display format

The Name column displays each process in the following format: `process_name(process_id): status`

where status is one of the following: Add, Delete, Referenced by code, Referenced by static, Grouped, Persistent, or RAM.

Status	Description
Add or Delete	This status appears when you perform a Compare to Snapshot to indicate new or removed objects since the last snapshot.
Referenced by code or Referenced by static	This status appears when a code (a local variable) or static data member references the variable.

Use the Objects tool to locate a memory leak

1. In the BlackBerry® Integrated Development Environment, on the **Debug** menu, click **Go**.
2. On the **Debug** menu, click **Break Now**.
3. On the **View** menu, click **Objects**.
4. In the objects pane, click **GC**.
5. In the objects pane, click **Snapshot**.
6. On the **Debug** menu, click **Continue**.
7. Perform operations in the BlackBerry Java® Application that do not increase the number of reachable objects. For example, create a new contact and then delete it.
8. On the **Debug** menu, click **Break Now**.
9. In the objects pane, click **GC**.
10. Click **Compare to Snapshot**.

The objects pane displays the number of objects deleted and added since the previous snapshot. If the number of objects added is not the same as the number of objects deleted, you might have an object leak. To narrow new objects, use the **Type**, **Process**, and **Location** filters located at the top of the objects pane.

11. To save the contents of the objects pane to a comma separated values (.csv) file, click **Save**.

Show references to or from an object

- > In the objects pane, right-click an object, and then click **Show References to** or **Show References From**.

The object view narrows to show only the objects that have references to or from this object. Use the **Forward** and **Back** buttons to move back and forth through the reference chain.

Right-click an object, and then click **Show Recursive References To @nnnnnnnn** to display all objects that reference the selected object. An object can indirectly display another object through it.



Note: This operation might take a long time to complete.

Show the source code or static

Double-click **Code referencing @nnnnnnnn** or **Static referencing @nnnnnnnn** line to display that code or static. Click **Forward** and **Back** to move back and forth through the reference chain.

View local variables

1. On the **View** menu, click **Locals**.
2. Perform one of the following tasks:

Task	Steps
View local variables and their current values in the context of the current thread.	> Click the Locals tab.
View local variable names and expressions at and around the executing line.	> Click the Auto tab.
View an expanded view of the current object (this).	> Click the This tab.
Evaluate expressions.	> Click the Watch tab.

View variable or expression information

Task	Steps
View a value for a variable.	<ol style="list-style-type: none"> 1. Point the insertion point at a variable. 2. Press the CTRL key and click a variable.
View a value for an expression.	<ol style="list-style-type: none"> 1. Press the CTRL key and click an expression.

View static data

The static data pane displays the static data members of the current class.

The following options are available when you right-click the static data pane:

- Change the display of the Value field.
- Set a watch on a variable.
- View the source code of a variable's defining class.
- When the BlackBerry® Java® Application modifies an item, stop the BlackBerry Java Application.

Evaluate (watch) Java expressions

The Watch pane enables you to specify variables and expressions to watch continuously while debugging your BlackBerry® Java® Application.

1. Right-click the Watch pane.
2. Perform one of the following tasks:

Task	Steps
Set a new watch.	> Click Add Watch .
Remove a watch.	> Click Delete .
Remove all watches.	> Click Delete All .

See "View threads" on page 241 for more information about viewing the format of threads appearing in the watch pane.

See "View the data members of a process" on page 242 for more information about viewing the format of processes appearing in the watch pane.

View threads

The threads pane displays all threads running on the BlackBerry® device. The most recent thread appears yellow.

To view the source code in the text edit pane, double-click a thread. The BlackBerry Integrated Development Environment marks the line in the source code that starts the thread with an arrow.

Thread format

The Thread column displays each thread in the following format:

name(pid): status

where:

- name is the name of the process that starts the thread
- pid is the ID of the process that starts the thread
- status is one of the following:

Status	Description
running	thread is running
sleeping	thread is calling <code>Thread.sleep()</code>
waiting for notify	thread is calling <code>Object.wait()</code>
acquiring lock	thread is executing a "synchronized" statement and is being forced to wait

Make a thread current

When you make a thread current, the Call Stack changes to display the calls for the thread. Other windows might display current information relating to the new current thread.

1. From a variable window, right-click a thread.
2. Select **Make thread current**.

Expand objects

In the threads pane, the following fields indicate the status of an object:

- thread that currently owns the object (Thread owning lock: @nnnnnnnn)
- thread that calls `Object.wait()` (Thread waiting for notify: @nnnnnnnn)
- thread that attempts to enter a synchronized block for the object (Thread acquiring lock: @nnnnnnnn)



Note: Threads, and objects whose threads are deadlocked, also display in the following panes: objects, local variables, watch, static data, processes and locks. To update the context of the selected thread in all the debugging panes, right-click in the threads pane, and then click **Make current**.

View the data members of a process

The processes pane lists all the processes that are currently running in the BlackBerry® Integrated Development Environment. You can expand each process to view its data members.

The Process column displays each process in the format: `process_name(process_id)`.

- > To view data members, in the **Process** column, expand a process.

View the call stack

The call stack pane displays the calling methods at the current point of execution.

View the source code of a calling method

1. Right-click a method.
2. Click **Show Definition**.

The source file appears in the Edit window at the line of code that implements the class of the selected item. All BlackBerry® Integrated Development Environment panes update to reflect the new context.



Note: The first calling method that appears in the call stack pane is located at the bottom of the call stack.

View event logs

The event log pane displays all exception messages that the BlackBerry® Integrated Development Environment produces when you run a BlackBerry Java® Application in the BlackBerry Device Simulator or on a BlackBerry device. To identify an error that has occurred, use the event log pane to view the source code that caused the error message.

View the source of a logging message

- > In the event log pane, on the **Build** tab, double-click the error message.

View classes

Select a subset of classes

Type the Class Name Prefix and press **ENTER**. For example, type `java.lang`. In the classes pane, classes that start with the string typed in the Class Name Prefix field appear.

1. In the classes pane, right-click a class.
2. Perform one of the following tasks:

Step	Task
Click Source code .	> Display the source code that implements the selected class.
Click Break when exception thrown .	> Set the BlackBerry® Integrated Development Environment to trigger a breakpoint when the code throws an object of the selected class.
Click Break on new object .	> Set the BlackBerry® IDE to trigger a breakpoint when an object of the selected class is instantiated.

View the methods in a class

The methods pane displays all methods in a class. In the classes pane, double-click a class. The methods pane updates to display all methods in the selected class.

Optimize source code using the BlackBerry Integrated Development Environment profiler tool

To optimize your code, use the profiler tool of the BlackBerry® Integrated Development Environment. The profiler tool displays the percentage of time spent in each code area to the current point of execution.



Note: To improve the quality of results when you run the profiler tool, exit other Microsoft® Windows® applications.

Set profile options

1. In the profile pane, click **Options**.
2. On the **General** tab, set the following options:

Drop-down list	Option	Description
Method attribution	Cumulative	The profiler tool calculates the time spent executing bytecode in a method and all methods that the method invokes.
	In method only	The profiler tool calculates the time spent executing bytecode in that method only. The timer stops when a call is made to another method.

Drop-down list	Option	Description
Sort method by	Count	The profiler tool sorts methods in the profile pane by the number of times the BlackBerry® Java® Application executed the item
	Profiled data (select in "What to profile")	The profiler tool sorts methods in the profile pane by the data you choose to profile
What to profile	Time (clock ticks)	The profiler tool considers execution time (measured in clock ticks).
	Number of objects created	The profiler tool considers the number of objects the BlackBerry Java Application created.
	Size of objects created	The profiler tool considers the size of objects the BlackBerry Java Application created.
	Number of objects committed	The profiler tool considers the number of objects the BlackBerry Java Application committed.
	Size of objects committed	The profiler tool considers the size of objects the BlackBerry Java Application committed.
	Number of objects moved to RAM	The profiler tool considers the number of objects the BlackBerry Java Application moved into memory.
	Size of objects moved to RAM	The profiler tool considers the size of objects the BlackBerry Java Application moved into memory.
	User Counting	The profiler tool considers user counting.

- Click the **Colors** tab to change the colors of the source code highlighting.

Generate profile data

- Set a breakpoint at the start of the section of code to profile.
- Set a breakpoint at the end of the section of code to profile.
- On the **Debug** menu, click **Go**.
- In the BlackBerry® Device Simulator, run the BlackBerry Java® Application. The debugger pauses the BlackBerry Java Application when it reaches the first breakpoint.
- On the **View** menu, click **Profile**.
- In the profile pane, click **Options**.
- Select the type of method attribution, a sorting method, and the type of information to profile.
- Click **OK**.
- In the profile pane, click **Clear**. This action removes the profiler data and sets the running time of Java® code to 0.
- On the **Debug** menu, click **Go**.
- In the BlackBerry Device Simulator, run the BlackBerry Java Application. The debugger pauses the BlackBerry Java Application when it reaches the second breakpoint.
- If the profile pane is not visible, on the **View** menu, click **Profile**.

13. In the profile pane, click **Refresh**. This action retrieves all accumulated profile data from the BlackBerry JVM. This action does not clear Profiler data, so running a BlackBerry Java Application again adds to the data.
 - Use profile views to view information about the section of code that you just ran.
14. Click **Save** to save the contents of the profile pane to a comma separated values (.csv) file.

View profile data

The profile pane has three views. Each view contains details about an item of execution (such as a method), the percentage of time the BlackBerry® Java® Application ran the item, and the number of times the application ran the item.



Note: To view all accumulated data, click **Refresh**.

1. On the **View** menu, click **Profile**.
2. Click one of the following tabs:

View	Description
Summary	<p>The Summary view displays general statistics about the system and the garbage collector process. It displays the percentage of time that the BlackBerry® JVM spends idle, executing code, and performing quick and full garbage collection process. The Percent column displays the percent of total BlackBerry JVM running time, including idle and collection time.</p>
Methods	<p>The Methods view displays a list of modules, sorted either by the information that you are profiling or by the number of times the BlackBerry Java Application executed each item.</p> <ul style="list-style-type: none"> • Expand the All item to see a list of all methods. • Expand a specific module to see only its methods. • Right-click a method, and then click Profile Source to view source lines in the Source view. • Right-click a method, and then click Show Source to view source code in the Edit window. <p>In this view, the Percent column displays the percentage of VM execution time only, not including idle and garbage collection time.</p>
Source	<p>The Source view displays the source lines of a single method. You can navigate through the methods that call, and are called by, that method.</p> <p>The Source view displays the following items:</p> <ul style="list-style-type: none"> • A list of callers to the method, including the number of times that they make the call and the total time spent on these calls • A list of source lines for the method and the total time spent on these lines <p>You can expand a source line to show individual bytecode.</p> <p>You can further expand any bytecode that corresponds to a method invocation to show the target(s) of the method invasion.</p> <p>Right-click a line and select Show Source to view the source code in the Edit window.</p> <p>Click Back and Forward to follow the history of methods that you have visited in the Source view.</p>

Analyze code coverage

The Coverage tools display a summary of code that has run.

Run the Coverage tool

1. Set two or more breakpoints in your code.
2. Run the BlackBerry® Java® Application to the first breakpoint.
3. On the **View** menu, click **Coverage**.
4. To reset information to 0, in the coverage pane, click **Clear**.
5. Run the BlackBerry Java Application to the next breakpoint.
6. To display the percentage of code that you ran since you clicked **Clear**, in the coverage pane, click **Refresh**.

The Coverage pane displays the percentage of code that you ran. It displays a nested view of packages, classes, and methods, with the percentage of code executed in each.

View source code

- > In the coverage pane, double-click a method.

Green bars in the source code indicate that the source code ran, and red bars in the source code indicate that the source code did not execute.



Note: When you use the ternary if-else operator, the coverage tool displays accurate but misleading results. For example, your code might include the following statement:

```
a ? b : c;
```

if "a" is always true, then "c" will never execute; however, the coverage tool displays the statement as covered.

You can work around this by rewriting the code to avoid the ternary operator, as shown in the following code:

```
if( a ) {
    b;
} else {
    c;
}
```

The short-circuit logical operators && and || exhibit the same behavior.

Approve HTTP connections

The BlackBerry® device includes built-in security to prevent third-party BlackBerry Java® Applications from sending or receiving data without the knowledge of the BlackBerry device user. When a third-party BlackBerry Java Application attempts to open a connection, a dialog box prompts the BlackBerry device user to turn the connection on or off. To test this functionality on the BlackBerry Device Simulator, turn on the security function of the BlackBerry Device Simulator.

This security feature causes network BlackBerry Java Applications to hang if a BlackBerry Java Application makes an HTTP connection from the main thread.

When the main thread handles connection requests, the UI cannot initiate a dialog box to prompt the BlackBerry device user to approve the connection attempt. The BlackBerry Java Application locks because the connection request cannot complete until the BlackBerry device user approves it.

To resolve this issue, you must put the network connection request on a separate thread from the main thread so that it does not interfere with the process of the main thread.

Start the BlackBerry email simulator

The BlackBerry® email simulator lets you send and receive email messages between the BlackBerry Device Simulator and an actual email account, without a BlackBerry Enterprise Server. To retrieve the BlackBerry email simulator, download the BlackBerry Email and MDS Services Simulator Package from the BlackBerry Developer Zone site:

www.blackberry.com/developers

1. On the **Start** menu, click **Programs > Research In Motion > BlackBerry Email and MDS Services Simulators 4.1.2 > ESS**.
2. Select one of the following modes:

Mode	Description
Standalone mode	ESS stores messages on the local file system and communicates directly with the email client. You do not require a POP3 or SMTP server. ESS can communicate with any email client that supports POP3 and SMTP communication. The email client account must have the POP3 server set to localhost on port 110 and the SMTP server set to localhost on port 25.
Connected mode	ESS polls the specified POP3 email server for incoming messages and uses the specified SMTP server to send messages. The Connected mode requires a Valid POP3 and SMTP server.

3. If you select Standalone mode, click **Clean FS** to erase ESS messages that are on the local file system.
4. If you select Connected mode, type information in the following fields:

Field	Description
Outgoing	host name of the SMTP server that your email account uses
Incoming	host name of the POP3 server that your email account uses
User name	user name with which to connect to your email account
Password	password with which to connect to your email account
Poll inbox	frequency with which the BlackBerry Device Simulator checks your email inbox for new messages

5. Type information in the following fields:

Field	Description
Name	name to display in outgoing messages from the BlackBerry Device Simulator
Email	email address to display in outgoing messages from the BlackBerry Device Simulator
PIN	PIN that the BlackBerry Device Simulator uses (default is 21000000)

6. Click **Launch**.
7. Click **Load Test** to select a message inside the associated **Inbox** and send it a number of times to a BlackBerry device.

 One or more email messages must exist inside the Inbox of the email account associated with the BlackBerry email simulator for the load test functionality to work.
8. Check the command prompt window for detailed information about ESS startup, including any login errors.

Working with compiled BlackBerry Java Applications

When you build a project using the BlackBerry® Integrated Development Environment, the BlackBerry IDE compiles your source files into Java® bytecode, performs preverification, and creates a single .cod file and .jad file for a BlackBerry Java Application.

If a BlackBerry Java Application contains more than 64 KB of bytecode or resource data, the BlackBerry IDE creates a .cod file that contains sibling .cod files. To determine if a .cod file contains sibling .cod files, extract the contents of the .cod file. Any .cod files within the original .cod file are the sibling files.

To identify modules that a BlackBerry Java Application requires, but are not provided with it, examine the BlackBerry Java Application descriptor (.jad) file RIM-COD-Module-Dependencies attribute. See “Appendix: BlackBerry Java Application .jad files” on page 271 for more information on BlackBerry Java Application .jad file properties.

Load and remove BlackBerry Java Applications

To load, remove, or save .cod files when testing BlackBerry® Java® Applications, use the JavaLoader.exe tool. For production applications, use the BlackBerry Desktop Software.



Note: You must load BlackBerry Java Applications with dependencies in the correct order. If project A is dependent on project B, load the project B .cod file before loading the project A .cod file.

1. Connect the BlackBerry device to the computer.
2. Open a command prompt, and navigate to the location of the JavaLoader.exe file.
3. Perform one of the following actions:

Task	Steps
Load a BlackBerry® Java® Application .cod file on the BlackBerry device.	<p>> Issue a command using the following format:</p> <pre>javaloader [-u] load .cod file</pre> <p>For example: javaloader.exe -u load MyApplication.cod</p> <p>JavaLoader loads the .cod files listed in the .jad file on the BlackBerry device and stores the .cod files in a CodeModuleGroup.</p>
Load BlackBerry® Java® Application .cod files listed in the same .jad file on the BlackBerry device.	<p>> Issue a command using the following format:</p> <pre>javaloader [-u] load .jad file</pre> <p>For example: javaloader.exe -u load MyApplication.jad</p> <p>JavaLoader loads the .cod files listed in the .jad file onto the BlackBerry device.</p>
Remove a BlackBerry® Java® Application .cod file from the BlackBerry device.	<p>> Issue a command using the following format:</p> <pre>javaloader [-u] erase [-f] module</pre> <p>For example: javaloader.exe -u erase MyApplication</p>
Remove BlackBerry® Java® Application .cod files listed in the same .jad file from the BlackBerry device.	<p>> Issue a command using the following format:</p> <pre>javaloader delete .jad file</pre> <p>For example: javaloader.exe delete MyApplication.jad</p>

Task	Steps
Remove BlackBerry® Java® Application .cod files stored in the same CodeModuleGroup from the BlackBerry device.	<p>> Issue a command using the following format:</p> <pre>javaloader delete [-g] module</pre> <p>For example: javaloader.exe delete -g MyApplication</p>
Save a BlackBerry® Java® Application .cod file from the BlackBerry device to your computer.	<p>> Issue a command using the following format:</p> <pre>javaloader save .cod file</pre> <p>For example: javaloader.exe save MyApplication.cod</p>
Save BlackBerry® Java® Application .cod files listed in the same .jad file from the BlackBerry device to your computer.	<p>> Issue a command using the following format:</p> <pre>javaloader save .jad file</pre> <p>For example: javaloader.exe save MyApplication.jad</p>
Save BlackBerry® Java® Application .cod files stored in the same CodeModuleGroup from the BlackBerry device to your computer.	<p>> Issue a command using the following format:</p> <pre>javaloader save [-g] module</pre> <p>For example: javaloader.exe save -g MyApplication</p>

View BlackBerry Java Application information

1. Connect the BlackBerry® device to the computer.
2. Open a command prompt and navigate to the location of the JavaLoader.exe file.
3. Issue one of the following tasks:

Task	Steps
Retrieve Name, Version, Size, and Date created information for a .cod file.	<p>> Issue a command using the following format:</p> <pre>javaloader info .cod file</pre> <p>For example: javaloader.exe info MyApplication.cod</p>
Retrieve a list of .cod files that a .cod file requires to run.	<p>> Issue a command using the following format:</p> <pre>javaloader info [-d] .cod file</pre> <p>For example: javaloader.exe info -d MyApplication.cod</p>
Retrieve information on <ul style="list-style-type: none"> • sibling .cod files • size of code section • size of data section • size of initialized data • number of class definitions • list of signatures applied to a .cod file 	<p>> Issue a command using the following format:</p> <pre>javaloader info [-v] .cod file</pre> <p>For example: javaloader.exe info -v MyApplication.cod</p>

Packaging and distributing BlackBerry Java Applications

Preverify BlackBerry Java Applications

Determine if your code requires signatures

Register to use RIM controlled APIs

Request code signatures

Distributing BlackBerry Java Applications over the wireless network

Distributing BlackBerry Java Applications with the BlackBerry Desktop Software

Preverify BlackBerry Java Applications

To reduce the amount of processing the BlackBerry® device performs when you load your BlackBerry Java® Application, partially verify your classes.

> Issue a command from the command line in the following format:

```
preverify.exe [-d] output -classpath directory input; directory
```

You may also use the BlackBerry Device Simulator to preverify .cod files. See the *BlackBerry Device Simulator User Guide* for more information about the BlackBerry Device Simulator.

Determine if your code requires signatures

Research In Motion (RIM) tracks the use of sensitive APIs in the BlackBerry® Java® Development Environment for security and export control reasons.

- > Locate the item in the API Reference. If the item has a lock icon or is noted as 'signed' your BlackBerry Java Application requires a signed key or signature, which RIM provides, before you can load the BlackBerry Java Application .cod files onto a BlackBerry device.

Controlled APIs

Three categories of Research In Motion (RIM) Controlled APIs exist: Runtime APIs, BlackBerry® Application APIs, and BlackBerry Cryptography APIs. See the *API Reference* for more information about all RIM controlled APIs.

You can run BlackBerry Java® Applications that use controlled APIs in the BlackBerry Device Simulator without code signatures; however, you must request code signatures from RIM before you can load these BlackBerry Java Applications on BlackBerry devices.



Note: To test and debug your code before receiving code signatures, use the BlackBerry Device Simulator. Code must have code signatures for deployment to BlackBerry devices.

If you use any of the following BlackBerry® API packages, your BlackBerry Java Application requires code signatures before you can load it on a BlackBerry device:

- `net.rim.blackberry.api.browser`
- `net.rim.blackberry.api.invoke`
- `net.rim.blackberry.api.mail`
- `net.rim.blackberry.api.mail.event`
- `net.rim.blackberry.api.menuitem`
- `net.rim.blackberry.api.options`
- `net.rim.blackberry.api.pdap`
- `net.rim.blackberry.api.phone`
- `net.rim.blackberry.api.phone.phonelogs`
- `net.rim.device.api.browser.field`
- `net.rim.device.api.browser.plugin`
- `net.rim.device.api.crypto.*`
- `net.rim.device.api.io.http`
- `net.rim.device.api.notification`
- `net.rim.device.api.servicebook`
- `net.rim.device.api.synchronization`
- `net.rim.device.api.system`

Register to use RIM controlled APIs

1. Complete the registration form on the BlackBerry® Developer Zone at <https://www.blackberry.com/JDEKeys>.
2. Save the .csi file that Research In Motion (RIM) sends to you in an email message. The .csi file contains a list of signatures and your registration information.

If the BlackBerry Signing Authority Tool administrator does not provide you with the .csi file or the Client PIN and you are an ISV partner, contact your ISV Technical Partnership Manager. If you are not an ISV partner, send an email message to jde@rim.com.

3. Double-click the .csi file.
4. If a dialog box appears that states that a private key cannot be found, follow the instructions to create a new key pair file.
5. In the **Registration PIN** field, type the **PIN** that RIM provided.

6. In the **Private Key Password** field, type a password of at least eight characters. The private key password protects your private key. If you lose this password, you must register again with RIM. If this password is stolen, contact RIM immediately.
7. Click **Register**.
8. Click **Exit**.

Restricted access to code signatures

The BlackBerry® Signing Authority Tool administrator might place restrictions on your .csi file to limit your access to code signatures. To request changes to these restrictions, contact your system administrator.

.csi file restriction	Description
# of Requests	<p>Sets the maximum number of requests you can make using a particular .csi file. When you reach the maximum number of requests, the .csi file becomes invalid. To make new code signature requests, you must apply for a new .csi file.</p> <p>Although an administrator can permit an infinite number of requests, the number of requests is often set to a finite number for security reasons.</p>
Expiry Date	Sets the expiry date for your .csi file. After the expiry date, you can no longer apply for code signatures with this .csi file. To make new signature requests, you must apply for a new .csi file.

Request code signatures

To perform this task, you must obtain a .csi file from Research In Motion (RIM). See “Register to use RIM controlled APIs” on page 252 for more information on obtaining a .csi file from RIM.

1. In Windows® Explorer, locate the .cod file for the BlackBerry® Java® Application for which you are requesting a signature.
2. Make sure that a .csi file with the same name as the .cod file exists in the same folder as the .cod file. The BlackBerry Integrated Development Environment compiler automatically generates the .csi file.
3. Double-click the .cod file to add it to the signature list. The signature list contains information on the .cod files that you want permission to access and are requesting signatures for.
4. Repeat steps 1 through 3 for each .cod file that you want to add to the signature list.
5. On the BlackBerry Signature Tool menu, click **Request**.

The BlackBerry Signature Tool is part of the BlackBerry Java® Development Environment installation. The BlackBerry JDE is available for download from the BlackBerry Developer Zone:

<http://www.blackberry.com/developers/>

6. In the dialog box, type your private key password.
7. Click **OK**. The BlackBerry Signature Tool uses the private key password to append the signature to the request, and it sends the signature list of .cod files to the Web Signer application for verification. The Web Signer application installs when you install the BlackBerry Signing Authority Tool. See the *BlackBerry Signing Authority Tool Version 1.0 - Password Based Administrator Guide* for more information about the Web Signer application.

Request code signatures using a proxy server

Task	Steps
Register signature keys using a proxy server.	<p>You can register each .csi file only once.</p> <ol style="list-style-type: none"> At the command prompt, browse to the BlackBerry® Signature Tool bin directory. For example: C:\Program Files\Research In Motion\BlackBerry JDE 4.3.1\bin Type the following command: <pre>Java -jar -Dhttp.proxyHost=myproxy.com -Dhttp.proxyPort=80 SignatureTool.jar SigKey.csi</pre> <ul style="list-style-type: none"> SigKey: The name of each signature key (.csi) file. Use the following naming conventions for the keys: client-RRT-*.csi, client-RBB-*.csi, client-RCR-*.csi. Dhttp.proxyHost: The name or IP address of the proxy server. Dhttp.proxyPort: The proxy server port number if you do not specify 80 as the default port number. Repeat step 2 for each .csi file that you want to register.
Sign a BlackBerry® Java® Application using a proxy server.	<ol style="list-style-type: none"> At the command prompt, browse to the BlackBerry Signature Tool bin directory. For example: C:\Program Files\Research In Motion\BlackBerry JDE 4.3.1\bin Type the following command: <pre>Java -jar -Dhttp.proxyHost=myproxy.com -Dhttp.proxyPort=80 SignatureTool.jar</pre> In the File Selection window, select the .cod file(s) to sign. Click Open.

Request a replacement registration key

Your registration key and .csk file are stored together. If you lose the registration key or the .csk file, you cannot request code signatures.

- > If you are an ISV partner and lose the .csk file, contact your ISV Technical Partnership Manager.
- > If you are not an ISV partner, send an email message to jde@rim.com.

View signature status

- Start the BlackBerry® Signature Tool.
- Select a .cod file.
- View the Status column:
 - For files the Web Signer has signed, the **Status** column contains **Signed**.
 - For files the Web Signer did not sign, the **Status** column contains **Failed**. The Web Signer might have rejected the .cod file because the private key password was typed incorrectly.

Distributing BlackBerry Java Applications over the wireless network

Method	Description
User-initiated wireless pull	Developers can post their compiled BlackBerry® Java® Applications to a public or private web site, and BlackBerry device users can download the BlackBerry Java Application over the wireless network by pointing the web browser on their BlackBerry devices to this URL. When a BlackBerry device user visits the URL, the browser prompts the BlackBerry device user to install the BlackBerry Java Application. If the BlackBerry device user accepts, the BlackBerry Java Application downloads over the wireless connection and installs immediately.
Server-initiated wireless push	In an enterprise environment, the BlackBerry® Enterprise Server administrator can push BlackBerry Java® Applications out to BlackBerry device users over the wireless network and enforce that the BlackBerry Java Application installs. The administrator simply creates a new policy and indicates that the BlackBerry Java Application is required. Once the policy is set on the BlackBerry Enterprise Server, the BlackBerry Java Application is sent to the BlackBerry device users without the need for any actions on the part of the BlackBerry device users. See the <i>BlackBerry Enterprise Server for Microsoft Exchange System Administration Guide</i> for more information about pushing BlackBerry Java Applications to a BlackBerry device over the wireless network.

Distribute applications using wireless pull

Deploying a .jad file with .cod files is the most efficient way to deploy BlackBerry® Java® Applications for a BlackBerry device using wireless pull, since no transcoding by either the BlackBerry device or the BlackBerry MDS Data Optimization Service feature of the BlackBerry Enterprise Server is required.

If you deploy the .jad file and .jar files for a BlackBerry Java Application, when a BlackBerry device user uses a BlackBerry device with the BlackBerry Enterprise Solution or the BlackBerry Internet Service to download the .jar file, the BlackBerry MDS Data Optimization Service feature of the BlackBerry Enterprise Server or the BlackBerry Internet Service transcodes the .jar file into a .cod file. If a BlackBerry device user uses a WAP browser to download the .jar file, the BlackBerry device transcodes the .jar file into a .cod file.

Task	Steps
Deploy .cod files.	<ol style="list-style-type: none"> Set the required MIME type on the web server. <ul style="list-style-type: none"> For .cod files, set the MIME type to application/vnd.rim.cod. For .jad files, set the MIME type to text/vnd.sun.j2me.app-descriptor. For .jar files, set the MIME type to application/java-archive. Place the .cod and .jad files on the web server for the BlackBerry® device users to download. BlackBerry device users use the BlackBerry device browser to browse to the .jad file and download the BlackBerry Java® Application. <p>Note: When you name .cod files, do not create a .cod file using the name-#.cod format, for example, my-1.cod, my-2.cod. If you use this format, the BlackBerry device recognizes the .cod file as sibling .cod files, and the .cod file does not properly load on to the BlackBerry device.</p>

Perform advanced BlackBerry Java Application distribution tasks

Task	Steps
Deploy .jar files.	<p>This task requires that a BlackBerry® device user download a .jar file to a BlackBerry device that connects to a BlackBerry Enterprise Server.</p> <ul style="list-style-type: none"> > Make .jar files available for download. When BlackBerry device users use the BlackBerry Browser to download a .jar file, the BlackBerry MDS Data Optimization Service feature of the BlackBerry Enterprise Server converts the .jar file to a .cod file.
Set .cod file dependencies.	<p>If any of the required modules are not present, the BlackBerry® Browser prevents the wireless installation of the BlackBerry Java® Application and lists the missing modules for the BlackBerry device user.</p> <ul style="list-style-type: none"> > In the BlackBerry Java Application descriptor (.jad) file, use the RIM-COD-Module-Dependencies attribute to specify modules that the BlackBerry Java Application requires, but that the BlackBerry Java Application does not provide. <p>For example, a BlackBerry Java Application that requires the RIM XML library might use the following in the BlackBerry Java Application descriptor: RIM-COD-Module-Dependencies: <code>net_rim_cldc, net_rim_xml</code></p>
Determine if a .cod file contains sibling .cod files.	<p>The following information is required only if BlackBerry® device users access BlackBerry Java® Applications using the BlackBerry Internet Service or a WAP gateway.</p> <ul style="list-style-type: none"> > Extract the contents of the .cod file. Any .cod files within the original .cod file are the sibling files.
Deploy a .cod file with sibling .cod files to a BlackBerry® device that is not connected to a BlackBerry Enterprise Server.	<ul style="list-style-type: none"> > Modify the BlackBerry Java® Application .jad file so that the file lists each sibling file individually.

Task	Steps
Extract sibling .cod files.	<p>To ensure a BlackBerry® device user does not override the original .cod file, on the web server, extract the sibling .cod files into a different directory than the directory where the original file exists.</p> <ol style="list-style-type: none"> 1. Unzip the original .cod file and extract the sibling .cod files. 2. Place each sibling .cod file on a web server. 3. In the .jad file, list the sibling .cod files separately. For each sibling file, create RIM-COD-URL -<#>, and RIM-COD-Size -<#> parameters. Use the following naming convention for sibling .cod files: <i>name of original .cod file - sequential number</i>. <ul style="list-style-type: none"> • RIM-COD-URL -<#>: Create a RIM-COD-URL -<#> parameter for each sibling .cod file, and place the name of the sibling file to the right of this parameter. # is a number that starts at 1 and increases by 1 for each sibling file. Give each sibling .cod files the same name as the original .cod file, followed by -<#>. • RIM-COD-Size -<#>: Create a RIM-COD-Size -<#> parameter for each sibling .cod file, and place the size of the sibling file to the right of this parameter. # is the same number that is appended to the name of the sibling file. Place the RIM-COD-Size -<#> parameter immediately below the RIM-COD=URL -<#> parameter. <p>The following example shows two sibling files. The developer names the sibling files myApp-1.cod and myApp-2.cod, after the original .cod file myAPP. The developer appends the '.cod' file extension to each sibling file name. The developer creates a RIM-COD-Size -<#> parameter for each sibling file.</p> <pre>Manifest-Version: 1.0 MIDlet-Version: 1.0.0 MIDlet-1: ., RIM-COD-Module-Dependencies: net_rim_cldc MicroEdition-Configuration: CLDC-1.0 RIM-COD-Module-Name: MyApp MIDlet-Name: My Application RIM-COD-URL: myApp.cod RIM-COD-Size: 55000 RIM-COD-URL-1: myApp-1.cod RIM-COD-Size-1: 50000 RIM-COD-URL-2: myApp-2.cod RIM-COD-Size-2: 25000 MicroEdition-Profile: MIDP-1.0</pre>
Distribute individual sibling .cod files.	Place each sibling .cod file onto a web server.

Modify information in a .jad file

You can use the Updatejad tool to process .jad files and perform the following actions:

- Correct the .cod file sizes listed in a .jad file. The .cod file sizes listed in the .jad file change after you use the BlackBerry® Signing Authority Tool to sign .cod files.
- Create .jad files that reference multiple .cod files.
- Add additional attributes to a .jad file. If you try to add an attribute that already exists in the .jad file, the existing attribute is not overwritten.

Use Updatejad only on .jad files created using the BlackBerry IDE or the RAPC command-line tool, and signed using the BlackBerry Signing Authority Tool. See the *BlackBerry Integrated Development Environment Help* or the *BlackBerry Signing Authority Tool Version 1.0 - Password Based Administrator Guide* for more information.

Updatejad commands have the following format:

```
updatejad.exe -q -n input.jad [additional.jad]
```

Options	Description
-q	Suppresses the creation of success output messages for .jad file processing operations. If an error occurs during .jad file processing, a non-zero exit code is produced.
-n	Suppresses the backup of the original .jad file.
input.jad	Specifies the .jad file to update.
additional.jad	Specifies other attributes to add to the input.jad file.

Task	Steps
Correct the .cod file sizes listed in a .jad file.	<ol style="list-style-type: none"> 1. Use the BlackBerry® IDE to create two BlackBerry Java® Application files, for example, test.cod and test.jad. 2. Use the BlackBerry Signing Authority Tool to sign test.cod. 3. From a command-prompt, navigate to the location of the Updatejad tool. 4. Type a command to correct the .cod file sizes listed in test.jad. For example, type updatejad.exe test.jad
Create .jad files that reference multiple .cod files.	<ol style="list-style-type: none"> 1. Use the BlackBerry® IDE to create two BlackBerry Java® Application files, for example, lib.cod and lib.jad. 2. Use the BlackBerry Signing Authority Tool to sign the .cod file. 3. Use the BlackBerry IDE to create two other BlackBerry Java Application files that use the .jad file, for example, test.cod and test.jad. 4. Use the BlackBerry Signing Authority Tool to sign the new .cod file. 5. From a command-prompt, navigate to the location of the Updatejad tool. 6. Type a command to correct the .cod file sizes listed in the new .jad file (in this example, test.jad) and to add the .cod file names from the first .jad file to the new one. For example, type, updatejad.exe test.jad lib.jad

Adding additional attributes to a .jad file

When multiple additional .jad files are provided, the first .jad file to contain an attribute determines the value associated with that attribute. The one exception is the RIM-COD attributes that include all values.

When additional .cod files are specified using one or more additional .jad files, they are added to the list of RIM-COD-URL-n tags in the order they are encountered.



Note: Once you load a BlackBerry® Java® Application onto a BlackBerry device, the BlackBerry Java Application cannot access any .jad file attributes added using the Updatejad tool.

Deployment of .cod files to a web server may require the extraction of siblings on the web server. The Updatejad tool does not extract the contents of sibling .cod files.

Distributing BlackBerry Java Applications with the BlackBerry Desktop Software

Deployment method	Description
Application Loader tool of the BlackBerry® Desktop Manager	The Application Loader tool of the BlackBerry Desktop Manager lets you install third party BlackBerry Java Applications as well as updated system software for the BlackBerry device. It lets BlackBerry device users download BlackBerry Java® Applications on their computers and install them on their BlackBerry devices.
BlackBerry® Application Web Loader	With the BlackBerry Application Web Loader, you can post your compiled BlackBerry Java® Application on a central web site and BlackBerry device users can install the BlackBerry Java Application by using Microsoft® Internet Explorer® on their computer to visit the URL. When BlackBerry device users visit the web page, they are asked to connect their BlackBerry device to the USB port. The BlackBerry Java Application is then installed using an ActiveX® Control. Application Web Loader provides a simple approach for installing BlackBerry Java Applications from your desktop and does not require the BlackBerry device user to run BlackBerry Desktop Manager. See the <i>BlackBerry Application Web Loader Developer Guide</i> for more information on the BlackBerry Application Web Loader.
Javaloader Command Line Tool	The BlackBerry® JDE includes a command line tool called Javaloader.exe. The executable file is in the BIN directory under the JDE installation. Javaloader can be used to quickly install and remove compiled BlackBerry Java® Application files on the BlackBerry device directly over the USB port and does not require any descriptor files or web pages. Javaloader can be useful when you are installing and removing your BlackBerry Java Application frequently during testing and development. However, Javaloader is not a tool that BlackBerry device users would use.

Create an application loader file

1. Create an .alx file for each BlackBerry® Java® Application, and then distribute the .alx file with the .cod files to BlackBerry® device users. See the *Application Loader Online Help* for more information about .alx files.
2. In the BlackBerry Integrated Development Environment, select a project.
3. On the **Project** menu, click **Generate .alx File**.

Load a BlackBerry Java Application on a specific BlackBerry device

1. Open a text editor.
2. Locate the .alx file for the BlackBerry® Java® Application.
3. In the .alx file, make sure the series attribute in the **fileset** opening tag refers to the BlackBerry device you want the BlackBerry Java Application to load on.

```
<fileset series="8700" Java="1.0">
```

For more information on the series attribute, see the Platform.alx file located in the simulator directory of your BlackBerry Java® Development Environment installation directory:

Program Files\Research In Motion\BlackBerry JDE 4.3\simulator.

4. Make sure the **files** tag contains a reference to the .cod file for your BlackBerry Java Application.

```
<files>
  My_application.cod
</files>
```

5. Update the **application**, **description**, and other tags to reflect the purpose of the .alx file.

```
<application id="Push only to 8700">
...
<description>This will push the COD only to 8700s</description>
```

Example: Load a BlackBerry® Java® Application on a specific BlackBerry device

```
<loader version="1.0">
  <application id="Push only to 8700">
    <name>Alien</name>
    <description>This will push the COD only to 8700s</description>
    <version>2006.02.14.1838</version>
    <vendor>RIM</vendor>
    <copyright>Copyright (c) 2001-2005</copyright>
    <fileset series="8700" Java="1.0">
      <files>
        My_application.cod
      </files>
    </fileset>
  </application>
</loader>
```

Specify optional components

In most cases, you do not need to change the .alx files that the BlackBerry® Integrated Development Environment generates.

1. Open a text editor.
2. In the text editor, open the .alx file you want to change.
3. To provide optional components for a BlackBerry Java® Application, in the .alx file, create a nested structure.

The .alx file uses an XML format:

Sample .alx file

```
<loader version="1.0">
  <application id="com.rim.samples.device.httptest">
    <name>Sample Network Application</name>
    <description>Retrieves a sample page over HTTP connection.
    </description>
    <version>1.0</version>
    <vendor>Research In Motion</vendor>
    <copyright>Copyright 1998-2003 Research In Motion</copyright>
    <language langid="0x000c">
      <name>Application D'Échantillon</name>
      <description>Obtenir une page du réseau
    </description>
    </language>
  </application>
</loader>
```

```

    <directory>samples/httpdemo</directory>
    <files>
        net_rim_httpdemo.cod
        net_rim_resource.cod
        net_rim_resource__en.cod
        net_rim_resource__fr.cod
    </files>
</fileset>
</application>
</loader>

```

4. To define an explicit dependency on another BlackBerry Java Application or library, use the `<requires>` tag.

Example: Sample .alx file for a BlackBerry® Java® Application with a nested module

```

<loader version="1.0">
  <application id="net.rim.sample.contacts">
    <name>Sample Contacts Application</name>
    <description>Provides the ability to store a list of contacts.
    </description>
    <version>1.0</version>
    <vendor>Research In Motion</vendor>
    <copyright>Copyright 1998-2001 Research In Motion</copyright>
    <fileset Java="1.0">
      <directory>samples/contacts</directory>
      <files>
        net_rim_contacts.cod
        net_rim_resource.cod
        net_rim_resource__en.cod
        net_rim_resource__fr.cod
      </files>
    </fileset>
    <application id="net.rim.sample.contacts.mail">
      <name>Sample Module for Contacts E-Mail Integration</name>
      <description>Provides the ability to access the messaging
        application</description>
      <version>1.0</version>
      <vendor>Research In Motion</vendor>
      <copyright>Copyright 1998-2001 Research In Motion</copyright>
      <fileset Java="1.0">
        <directory>samples/contacts</directory>
        <files>
          net_rim_contacts_mail.cod
        </files>
      </fileset>
    </application>
  </application>
</loader>

```

Specify supported BlackBerry Device Software

BlackBerry® Java® Applications that use APIs only available on particular versions of the BlackBerry Device Software should specify supported BlackBerry device versions using the `_blackberryVersion` attribute.

- > Specify a range using the following rules:

- Square brackets [] indicate inclusive (closed) range matching.
- Round brackets () indicate exclusive (open) range matching.
- Missing lower ranges imply 0.
- Missing upper ranges imply infinity.

For example, [4.0,) indicates any version between 4.0 and infinity.

The following example prevents modules from loading on versions of the BlackBerry Device Software earlier than version 4.0.

```
<application id="application_id" _blackBerryVersion="[4.0,)">
...
</application>
```

The following example provides alternative modules for different versions of the BlackBerry Device Software.

```
<application id="application_id">
...
  <fileset _blackBerryVersion="(,4.0)">
    ... modules for BlackBerry device software versions earlier than 4.0
  </fileset>
  <fileset _blackBerryVersion="[4.0,)">
    ... modules for BlackBerry device software versions 4.0 and later
  </fileset>
</application>
```

See "Appendix: .alx files" on page 267 for more information about .alx file elements.

Appendix: The command line compiler

Using the command line compiler

Using the command line compiler

The BlackBerry® Java® Development Environment includes RAPC, a command line compiler. RAPC compiles .java and .jar files into .cod files that you can run in the BlackBerry Device Simulator or load onto a BlackBerry device.

The rapc.exe file exists in the **bin** subdirectory of your BlackBerry JDE installation.



Note: `net_rim_api.jar` is required as an input file when you invoke RAPC. Use the command line argument with the `-import=` option to provide this .jar file to RAPC.

RAPC accepts the following command line options in the following order:

Option	Option format	Description
<code>java class jar</code>		The input files: <ul style="list-style-type: none"> .java: A Java source program file that javac must compile. .class: A Java .class file that javac has compiled. .jar: An archive of files that the compilation set must include.
<code>jad</code>		An input file that contains BlackBerry Java Application information. For example, it contains a list of attributes as specified by the MIDP specification.
<code>-class</code>	<i>classname</i>	The name of the class containing the BlackBerry Java Application main entry point; without this option, RAPC uses the first <code>main (String[])</code> method it finds as the entry point.
<code>-codename</code>	<code>=[path\[...]]filename</code>	Specify the name and location of the output .cod file; typically the output .cod file uses the same name as the .jar file.
<code>-library</code>	<code>=[path\[...]]filename</code>	Specify the name and location of the output .cod file as a library.
<code>-import</code>	<code>=file.jar[;...]</code>	List dependent .jar files; for example list RIM APIs and other dependent libraries.
<code>-midlet</code>		Specify the .cod file as a MIDlet and generate a preverified .jar file.
<code>-deprecation</code>		The java compiler ignores the value specified with the -deprecation option.
<code>-nowarn</code>		The java compiler does not issue warnings for the value specified with the -nowarn option.
<code>-quiet</code>		Display only errors.
<code>-warning</code>		Generate warning messages.
<code>-verbose</code>		Display information about RAPC activity. RAPC stores this information in intermediate and temporary files in the BlackBerry device user's temporary folder. RAPC does not delete the temporary files.

Option	Option format	Description
-wx		Treat certain warnings as errors.
-warnkey	=0xNNNNNNNN[...]	Generate a warning if you need to add a key to the .csl file.
-workspace	= <i>filename</i>	Add the <i>filename</i> to the .debug file for BlackBerry Integrated Development Environment browsing.
filename_1.java [<i>additional .java files</i> <i>as required</i>]		Specify the .java file name if you are compiling from java files.
JAR_filename.jar		Specify the .jar file name if compiling from a .jar file.



Note: If you specify both the **-codename** and **-library** options, RAPC uses the **-library** option. For option values that start with an '=' symbol (for example: **-workspace**), the '=' before the option name is optional.

For example, the following command line instruction compiles the SampleApp.jar file into a .cod file of the same name:

```
rapc import=net_rim_api.jar codename=SampleApp\SampleAppDriver -midlet SampleApp.jad
Samples\SamplaApp.jar
```

Appendix: XML control entity attributes

Using XML control entity attributes

Using XML control entity attributes

Use the PAP DTD to specify the following attributes:

Goal description	XML control entity attributes	Example
Specify the equivalent of the REQUEST URI HTTP parameter for RIM push.	X-Wap-Application-Id	"/"
Specify a unique message ID. Additionally, use this control entity attribute to cancel or check the status of a message. Use a URL in combination with a value. For example, 123@blackberry.com.	push-id	123@wapforum.org
Specify the URL that the result notification is sent to.	ppg-notify-requested-to	http://wapforum:8080/ReceivePAPNotification
Specify the date and time by which to deliver the content to the BlackBerry® device. Content that has not been sent by this date and time is not delivered. Represent the date and time in UTC format: YYYY-MM-DDThh:mm:ssZ where <ul style="list-style-type: none"> • YYYY is a 4-digit year • MM is a 2-digit month • DD is a 2-digit day • hh is a 2-digit hour based on 24-hour timekeeping system • mm is a 2-digit minute • ss is a 2-digit second • Z indicates that the time is in UTC 	deliver-before-timestamp	2004-01-20T22:35:00z
Specify the date and time after which content is delivered to the BlackBerry® device. Content is not delivered before this date. Represent the date and time in UTC format.	deliver-after-timestamp	2004-01-20T21:35:00z
Specify the address of the BlackBerry device that the push content is sent to. The <i>destination</i> is the destination internet messaging address or PIN.	address-value	WAPPUSH=destination%3Aport//TYPE=USER@blackberry.com
Specify the delivery reliability mode of the content, transport-level, or application-level.	delivery-method	confirmed; unconfirmed

For more information about writing server-side push applications using PAP, see the *Push Access Protocol (WAP-247-PAP-20010429-a)* specification at <http://www.wmlclub.com>. See the *PAP 2.0 DTD* for information about the WAP Push DTDs.

Appendix: .alx files

Elements in BlackBerry application .alx files

Elements in BlackBerry application .alx files

Element	Attributes	Description
application	id	<p>The <code>application</code> element contains the elements for a single BlackBerry® Java® Application.</p> <p>The <code>application</code> element can also contain additional nested <code>application</code> elements. Nesting lets you require that, when a BlackBerry Java Application loads onto the BlackBerry device, its prerequisite modules also load onto the BlackBerry device.</p> <p>The <code>id</code> attribute specifies a unique identifier for the BlackBerry Java Application. To provide uniqueness, use an ID that includes your company domain in reverse order. For example, <code>com.rim.samples.docs.helloworld</code>.</p>
copyright	—	The <code>copyright</code> element provides copyright information, which appears in the Application Loader.
description	—	The <code>description</code> element provides a brief description of the BlackBerry Java Application, which appears in the Application Loader.
directory	—	<p>The <code>directory</code> element provides the location of a set of files. The <code>directory</code> element is optional. If you do not specify a <code>directory</code>, the files must exist in the same location as the .alx file. The <code>directory</code> element specifies the directory relative to the location of the .alx file.</p> <p>Directory elements are cumulative within a BlackBerry Java Application.</p> <p>For example:</p> <pre><application id="com.abc.my.app"> <directory>MyCodFiles</directory> <fileset Java="1.0"> <files> a.cod //resolves to <.alx location>\MyCodFiles b.cod </files> </fileset> <directory>MyCodFiles</directory> <fileset Java="1.0"> <files> c.cod //resolves to <.alx location>\MyCodFiles\MyCodFiles d.cod </files> </fileset> </application></pre>
files	—	The <code>files</code> element provides a list of one or more BlackBerry Java Application .cod files, in a single directory, to load onto the BlackBerry device.

Element	Attributes	Description
fileset	Java radio langid Colour	<p>The <code>fileset</code> element includes an optional <code>directory</code> element and one or more <code>files</code> elements. It specifies a set of .cod files, in a single directory, to load onto the BlackBerry device. To load files from more than one directory, include one or more <code>fileset</code> elements in the .alx file.</p> <p>The <code>Java</code> attribute specifies the minimum version of the BlackBerry JVM with which the .cod files are compatible. The current BlackBerry JVM is Version 1.0. The <code>Java</code> attribute is required.</p> <p>The <code>radio</code> attribute lets you load different BlackBerry Java Applications or modules onto the BlackBerry device depending on the network type of the BlackBerry device. Possible values include <code>Mobitex</code>, <code>DataTAC</code>, <code>GPRS</code>, <code>CDMA</code>, and <code>IDEN</code>. The <code>radio</code> attribute is optional.</p> <p>The <code>langid</code> attribute lets you load different BlackBerry Java Applications or modules depending on the language support that BlackBerry device users add to the BlackBerry device. The value is a Win32 <code>langid</code> code; for example: <code>0x0009</code> (English), <code>0x0007</code> (German), <code>0x000a</code> (Spanish), and <code>0x000c</code> (French). The <code>langid</code> attribute is optional.</p> <p>The <code>colour</code> attribute lets you load different BlackBerry Java Applications or modules for color or monochrome displays. The value is a Boolean; <code>true</code> means color display and <code>false</code> means monochrome.</p>
hidden	—	<p>The <code>hidden</code> element hides a package so that it does not appear to BlackBerry device users in the Application Loader. To hide a package, add the following line: <code><hidden>true</hidden></code>.</p> <p>Use this element in conjunction with the <code>required</code> element to load the BlackBerry Java Application by default, or set the <code>requires</code> tag to load this package if another BlackBerry Java Application exists.</p> <p>Only corporate system administrators should use the <code>hidden</code> tag. This tag is not intended for use by third-party software vendors.</p> <p>Note: The BlackBerry Desktop Software Version 3.6 or later supports this element.</p>
language	langid	<p>The <code>language</code> tag lets you override the text that appears in the Application Loader when the Application Loader runs in the language that the <code>langid</code> attribute specifies.</p> <p>To support multiple languages, specify multiple <code>language</code> tags. To specify the name, description, version, vendor, and copyright tags for each language, nest these tags in the <code>language</code> tag. If you do not nest a tag, the text appears in the default language.</p> <p>The <code>langid</code> attribute specifies the Win32 <code>langid</code> code for the language to which this information applies. For example, some Win32 <code>langid</code> codes are: <code>0x0009</code> (English), <code>0x0007</code> (German), <code>0x000a</code> (Spanish), and <code>0x000c</code> (French).</p>
library	id	<p>You can use the <code>library</code> element instead of the <code>application</code> element. It contains the elements for a single library module. You cannot nest modules. By default, a library module does not appear in the Application Loader.</p> <p>Typically, use the <code>library</code> element as the target of a <code><requires></code> element, so that when a particular BlackBerry Java Application loads onto the BlackBerry device, a required library also loads onto the BlackBerry device.</p> <p>Note: The BlackBerry Desktop Software Version 3.6 or later supports this element.</p>
loader	version	<p>The <code>loader</code> element contains one or more <code>application</code> element.</p> <p>The <code>version</code> attribute specifies the version of the Application Loader.</p>
name	—	<p>The <code>name</code> element provides a descriptive name for the BlackBerry Java Application, which appears in the Application Loader.</p>

Element	Attributes	Description
<code>required</code>	—	<p>The <code>required</code> element lets you force an application to load. The Application Loader selects the BlackBerry Java Application for installation, and the BlackBerry device user cannot change this selection. Add the following line: <code><required>true</required></code>.</p> <p>Only corporate system administrators should use the <code>required</code> tag. This tag should not be used by third-party software vendors.</p> <p>Note: The BlackBerry Desktop Software Version 3.5 or later supports this element.</p>
<code>requires</code>	<code>id</code>	<p>The <code>requires</code> element is an optional element that specifies the <code>id</code> of a package on which this BlackBerry Java Application depends. This element can appear more than once, if the BlackBerry Java Application depends on more than one BlackBerry Java Application. When a BlackBerry Java Application loads onto the BlackBerry device, all packages that the <code><requires></code> tag specifies also load onto the BlackBerry device.</p> <p>Note: The BlackBerry Desktop Software Version 3.6 or later supports this element.</p>
<code>vendor</code>	—	<p>The <code>vendor</code> element provides the name of the company that created the BlackBerry Java Application, which appears in the Application Loader.</p>
<code>version</code>	—	<p>The <code>version</code> element provides the version number of the BlackBerry Java Application, which appears in the Application Loader. This version number is for information purposes only.</p>

Appendix: BlackBerry Java Application .jad files

Properties of BlackBerry Java Application .jad files
Accessing application attribute properties from a .jad file

Properties of BlackBerry Java Application .jad files

Required RIM attribute	Description
RIM-COD-Creation-Time	creation time of the .cod file
RIM-COD-Module-Dependencies	list of modules that the .cod file requires
RIM-COD-Module-Name	name of the module that the .cod file contains
RIM-COD-SHA1	SHA1 hash of the .cod file
RIM-COD-Size	size (in bytes) of the .cod file
RIM-COD-URL	URL from which the .cod file can be loaded

Optional RIM attribute	Description
RIM-Library-Flags	reserved for use by Research In Motion (RIM)
RIM-MIDlet-Flags	reserved for use by RIM
RIM-MIDlet-NameResourceBundle	name of the resource bundle on which the BlackBerry® Java® Application depends
RIM-MIDlet-Position	suggested position of the BlackBerry Java Application icon on the Home screen Note: This position might not be the actual position of the BlackBerry Java Application icon on the Home screen.

The BlackBerry® Integrated Development Environment lets you create a dual-purpose .jad file to support the downloading of MIDlets onto BlackBerry devices and other wireless devices. To do this, create a .jad file that contains both the RIM-COD-URL and RIM-COD-Size attributes and the MIDlet-Jar-URL and MIDlet-Jar-Size attributes. On BlackBerry devices, download the .cod files; on other wireless devices, download the .jar files.

Set .cod file dependencies

The application descriptor (.jad) file contains a RIM-COD-Module-Dependencies attribute that specifies the modules that the BlackBerry® Java® Application requires, but are not provided with it. The RIM-COD-Module-Dependencies attribute lets a BlackBerry device user avoid downloading a BlackBerry Java Application that will not run. The RIM-COD-Module-Dependencies attribute takes a comma-separated module name list as a parameter. For example, a BlackBerry Java Application that requires the RIM XML library might use the following entry in the application descriptor:

```
RIM-COD-Module-Dependencies: net_rim_cldc, net_rim_xml
```

Accessing application attribute properties from a .jad file

After a BlackBerry device user uses the .jad file of a BlackBerry device application to download the application onto a BlackBerry device, the application can use the `CodeModuleGroup` to access the name-value pair information in the application's .jad file.

Let a BlackBerry device application retrieve name-value pair information from a .jad file

1. Load a code module group into a `CodeModuleGroup` object using the name of the required module group as a parameter, and store a reference to the `CodeModuleGroup` object. In this example, *group_name* refers to the name of the code module group.

```
CodeModuleGroup cmg = CodeModuleGroupManager.load(group_name);
```

2. Retrieve properties from the `CodeModuleGroup` by invoking the `CodeModuleGroup.getProperty(name)` method. The following example iterates through all the properties of a code module group.

```
CodeModuleGroup cmg = CodeModuleGroupManager.load(group_name);
for( Enumeration e = cmg.getPropertyNames(); e.hasMoreElements(); ) {
    String name = (String)e.nextElement();
    System.out.println( "Name: " + name );
    System.out.println( "Value: " + cmg.getProperty(name) );
}
```

See the API reference for the BlackBerry Java Development Environment for more information about the `CodeModuleGroup` class.

Appendix: The Eclipse development environment

Use the Eclipse development environment

Use the Eclipse development environment

The Java® Debug Wire Protocol (JDWP) program provides an interface to the BlackBerry® Device Simulator. When you start the JDWP, you can use third-party integrated development environments.

Start the JDWP

> Click **Start** > **Programs** > **Research In Motion** > **BlackBerry JDE 4.3** > **JDWP**.



Note: You must start the BlackBerry Device Simulator from the BlackBerry® IDE before you can start the JDWP. To start a BlackBerry Device Simulator in the Eclipse development environment, click **Run** > **Debug**.

Connect to the Eclipse development environment



Note: Before completing the tasks in this section, install and configure the Eclipse™ development environment.

Perform the following steps:

1. Extend the Sun JRE
2. Add the API documentation
3. Set builder settings
4. Set project variables

Extend the Sun JRE

1. Set up your workspace and project.
2. Start the Eclipse workbench
3. On the Eclipse taskbar, click **Window** > **Preferences**.
4. Expand the **Java** item.
5. Select **Installed JREs**.
6. Click **Add**.
7. In the Add JRE window, in the **JRE type** field, specify **Standard VM**.

8. In the **JRE name** field, type a name for the JRE.
9. In the **JRE home directory** field, type the location of the Sun JRE. For example:
C:\Java\jdk1.5.0_02\jre.
10. Make sure the **Use default system libraries** field is cleared.
11. Click **Add External JARs**.
12. Browse to the **lib** folder of your installation of the BlackBerry® JDE. For example:
C:\Program Files\Research In Motion\BlackBerry JDE 4.3\lib
13. Select **net_rim_api.jar**.
14. Click **Open**.

Add the API documentation

1. Add a RIM .jar file to your project.
2. In the Add JRE window, expand the **net_rim_api.jar** file.
3. Select **Javadoc location**.
4. Click **Edit**.
5. Click **Browse**.
6. Navigate to the **docs\api** folder of your BlackBerry® JDE installation. For example:
C:\Program Files\Research In Motion\BlackBerry JDE 4.3\docs\api
7. Click **OK**.
8. Click **OK**.
9. In the Installed JREs window, select the newly created JRE. The default is RIM JVM.
10. In the Add JRE window, click **OK**.

Set builder settings

1. On the **Eclipse** taskbar, click **Project > Properties**.
2. Select **Builders**.
3. Click **New**.
4. In the Choose configuration type window, select **Program**.
5. Click **OK**.
6. In the Properties for New_Builder window, in the **Name** field, type a name for the builder.
7. In the **Location** field, click **Browse File System**.
8. Navigate to the **bin** folder of your BlackBerry® JDE installation. For example:
C:\Program Files\Research In Motion\BlackBerry JDE 4.3\bin
9. Select the **rapc.exe** file.

10. Click **Open**.

Set project variables

1. In the **Working Directory** field, click **Variables**.
2. In the Select Variable window, select **Build project**.
3. Click **OK**.
4. In the **Arguments** field, type:

```
-quiet [desired space separated java, class, jar, or jad files] import="C:\Program Files\Research In Motion\BlackBerry JDE 4.3\lib\net_rim_api.jar" codename=C:\Development\ProjectName
```

Example:

```
-quiet C:\Development\TestProject\*.java import="C:\Program Files\Research In Motion\BlackBerry JDE 4.3\lib\net_rim_api.jar" codename=C:\Development\TestProject.
```

5. Click **OK**.
6. In the Properties for New_Builder window, click the **Build Options** tab.
7. In the **Run the builder** section, verify that the following options are selected:
 - **After a "Clean"**
 - **During manual builds**
 - **During auto builds**
8. Click **OK**.
9. In the Properties for window, click **OK**.



Note: RAPC does not support wildcard characters. If an input path error occurs, use a space separated list of files. For example, replace `C:\Development\TestProject*.java` with `C:\Development\A.java C:\Development\B.java`.

Set the connection time

To prevent connection timeouts when debugging in the Eclipse development environment, set the timeout values for the debug program.

1. On the Eclipse taskbar, click **Windows > Preferences**.
2. Expand the **Java** item.
3. Select **Debug**.
4. In the **Communication** section, in the **Debugger timeout** field, type a value.
5. In the **Launch timeout** field, type a value.



Note: The values you set in the **Debugger timeout** and **Launch timeout** fields depend on the processing speed of your computer. If connection problems continue after setting these fields, increase the timeout values.

Debug an application using the Eclipse development environment

1. From the Eclipse taskbar, click **Run > Debug**.
2. Select **Remote Java Application**.
3. Click **New**.
4. Click the **Source** tab.
5. Verify that your application is listed.
6. Click **Close**.
7. Open the **JDWP** application. See “Start the JDWP” on page 273 for more information.
8. On the Eclipse taskbar, click **Run > Debug**.
9. Under the **Remote Java Application** item, select an application.
10. Click **Debug**.



Note: If the following error message appears: “Failed to connect to remote VM. Connection timed out”, increase the debugger timeout values. See “Set the connection time” on page 275 for more information.

Acronym list

3

3GPP

3rd Generation Partnership Project

A

AES

Advanced Encryption Standard

API

application programming interface

APN

Access Point Name

ATR

Answer To Reset

B

bpm

beats per minute

C

CAC

common access card

CDMA

Code Division Multiple Access

D

DTD

document type definition

DTMF

Dual Tone Multiple Frequency

E

EDGE

Enhanced Data Rates for Global Evolution

EVDO

Evolution Data Optimized

G

GAN

generic access network

GERAN

GSM-EDGE Radio Access Network

GPRS

General Packet Radio Service

GPS

Global Positioning System

GSM

Global System for Mobile Communications

H

HTTP

Hypertext Transfer Protocol

HTTPS

Hypertext Transfer Protocol over Secure Sockets Layer

I

ID

identification

J

JSR

Java Specification Request

L

LAN

local area network

LBS

location based services

LMM

Low Memory Manager

M

MIDP

Mobile Information Device Profile

P

PAP

Push Access Protocol

PDE

Position Determination Entity

PME

Plazmic Media Engine

POST

power-on self test

R

RAM

random access memory

S

SSID

service set identifier

S/MIME

Secure Multipurpose Internet Mail Extensions

T

TCP

Transmission Control Protocol

TCP/IP

Transmission Control Protocol/Internet Protocol

Triple DES

Triple Data Encryption Standard

U

UDP

User Datagram Protocol

UI

user interface

UMA

Unlicensed Mobile Access

UMTS

Universal Mobile Telecommunications System

URI

Uniform Resource Identifier

USB

Universal Serial Bus

UTC

Coordinate Universal Time

UTRAN

UMTS Terrestrial Radio Access Network

W

WAP

Wireless Application Protocol

WLAN

wireless local area network

X

XML

Extensible Markup Language

